Escuela Superior de Ingeniería

ESCUELA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE CÁDIZ

Estudio de las redes definidas por software (SDN) y desarrollo de un prototipo para la Diputación de Cádiz

---

Autor: Javier Barroso Canto

Cádiz, julio 2018

Escuela Superior de Ingeniería

ESCUELA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE CÁDIZ

Estudio de las redes definidas por software (SDN) y desarrollo de un prototipo para la Diputación de Cádiz

---

Director: Carlos Rogríguez Cordón

Autor: Javier Barroso Canto

Cádiz, julio 2018

Agradecimientos

A Carlos, por proponerme y tutorizar cuidadosamente este proyecto e involucrarse en mi crecimiento académico.

A Manolo, por enseñarme cómo es el mundo de las redes más allá de clases y creer en mi trabajo; así como a los becarios y resto de trabajadores de EPICSA por hacer de esta, una experiencia superlativa.

A mis amigos y compañeros de facultad, por los buenos momentos que quedarán para el recuerdo; y en especial a Carlos, Jorge y Salva por coincidir en la mayoría de mis trabajos y no aburriros de mí.

A los tres Barroso que han aguantado con paciencia desmesurada mis malos momentos y han estado a mi lado siempre.

A las tres mujeres de mi vida, que con su amor han hecho de mí quien soy hoy en día.

Y a mi abuelo, que siempre creyó en mí.

Índice general

Índice de figuras

Índice de tablas

Estudio de las redes definidas por software (SDN) y desarrollo de un prototipo para la Diputación de Cádiz

Memoria

CLIENTE	EMPRESA PROVINCIAL DE INFORMACIÓN DE CÁDIZ, S.A. (EPICSA)
	PLAZA MADRID S/N, EDIFICIO CARRANZA, FONDO SUR, LOCAL 10, 11010 CÁDIZ
	956 26 15 00
SUMINISTRADOR	JAVIER BARROSO CANTO
	INGENIERO INFORMÁTICO
	49564855-Q
	JAVIER.BARROSOCANTO@ALUM.UCA.ES

FIRMA DEL CLIENTE **FIRMA DEL SUMINISTRADOR**

Cádiz, julio 2018

Memoria

Índice

1.1. Introducción

Las Redes Definidas por Software o `Software-Defined Networking` (SDN) constituyen una nueva arquitectura de red dispuesta como un nuevo paradigma capaz de solventar la demanda creciente de servicios, tales como redes sociales o correos electrónicos, donde arquitecturas de redes convencionales se están viendo superadas en los últimos años a causa de un gran auge en la cantidad de servicios que los usuarios demandan.

Este documento está estructurado para presentar toda información considerada relevante sobre el diseño e implementación de una aplicación SDN, capaz de gestionar la red de la diputación de Cádiz, que trabaje con un controlador que consiga imbuir determinados comportamientos en los nodos de la misma. Esta documentación seguirá la norma UNE 157801:2007 para la aplicación de criterios generales para la elaboración de proyectos de sistemas de información.

1.2. Objeto

El objetivo de este proyecto es estudiar, analizar y simular varios prototipos de arquitecturas SDN para, una vez estudidada todas las alternativas, seleccionar una de ellas e implementar una aplicación SDN en la Empresa Provincial de Información en Cádiz, S.A. (EPICSA).

Su finalidad es servir como base de conocimiento sobre la cual basar la toma de decisiones en cuanto a una implementación real de SDN en su red en un futuro.

1.3. Antecedentes

1.3.1. Estado actual de la entidad

Tal y como se definen en su propia web, la Empresa Provincial de Información de Cádiz, S.A. «es una empresa pública creada en 1984 por la Diputación de Cádiz con el objeto social de dar asistencia técnica informática integral, formación, comercialización y desarrollo e implantación de aplicaciones informáticas, tanto a la propia Diputación como a sus organismos, empresas y municipios de menos de 20.000 habitantes.» [6]

1.3.2. Modelo de negocio y actividad que desarrolla

EPICSA centra su actividad en garantizar el funcionamiento de las redes de datos y de las aplicaciones corporativas en el ámbito que le es de su competencia a nivel provincial. Por consiguiente, una de las actividades en la que está actualmen-

te trabajando es en el desarrollo de la e-Administración o Administración pública electrónica. [6]

1.3.3. Organigrama

La Empresa Provincial de Información de Cádiz se rige por el organigrama mostrado en la Figura 1.1. Se presenta, a continuación, una relación de los departamentos de la empresa [6]:

- Gerencia
- Administración
- Área de Informática de Gestión
  - Servicios Municipales Web y Multimedia
  - Servicios de Tramitación y Administración Electrónica
  - Servicios Corporativos
    - Servicios Generales
    - Recursos Humanos
  - Servicios Corporativos Especializados
    - Recaudación y Servicios Especiales
    - Nóminas y Control de Presencia
    - Comunicación e Innovación
- Área de Sistemas, Redes, CAU y Explotación
  - Servicio de Sistemas
  - Servicio de Redes
  - Servicio de CAU y Explotación

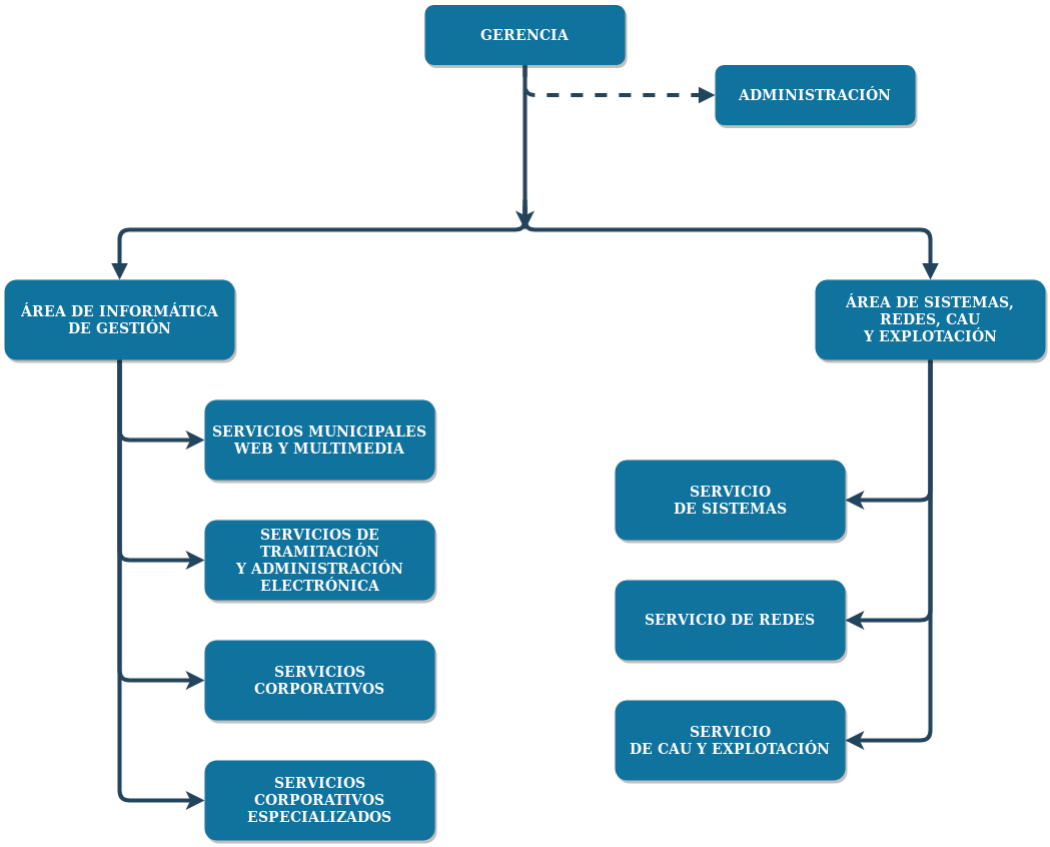

Figura 1.1: Organigrama de EPICSA

1.3.4. Ubicación de los departamentos en los inmuebles

Entre las instalaciones de las que dispone EPICSA, se encuentran:

- Planta baja
  - Área de Sistemas, Redes, CAU y Explotación.
  - Un Centro de Procesasamiento de Datos (CPD) y Servidores.
  - Un áula de formación.
  - Almacén.
  - Taller de reparación de equipos.
- Primera planta
  - Administración.
  - Área de Informática de Gestión.

1.4. Descripción de la situación actual

1.4.1. Evolución de las redes

Podemos dividir la historia de `Internet` en tres etapas:

- **Redes activas (1995 - 2000)**: El despliegue de `Internet` supuso multitud de sustituciones de aplicaciones por otras más novedosas. El aumento en el uso de estas aplicaciones provocó que los investigadores desarrollaran e implementaran nuevos protocolos de red, de forma que se pudiera gestionar y administrar los diferentes nodos de la red individualmente. [8]
- **Separación del plano de control del plano de datos (2001 - 2007)**: A partir del año 2000, debido a los fuertes aumentos en el volumen de tráfico, se vuelve a examinar y estructurar las arquitecturas tradicionales. Años más tarde, se resuelve mediante la separación del plano de control y el plano de datos en los `routers` y `switches` que componen la red; tal y como se muestra en la Figura 1.2 [25]
- **Aparición de la API de OpenFlow (2007 - 2010)**: En los últimos años, un grupo de investigadores de la universidad de Stanford junto a otros de `Hewlett Packard` (HP) crearon el proyecto `Ethane`, donde los administradores de red serían los únicos que podrían decidir qué nodo hablaba con qué otro nodo y por qué canal tenía que hacerse. Este proyecto acabaría convirtiéndose finalmente en el predecesor directo del protocolo OpenFlow, el cual complementa las ideas del proyecto `Ethane` con las tablas de flujo denominadas `Ternary Content-Addressable Memory` (TCAM) de los dispositivos convencionales. [15]

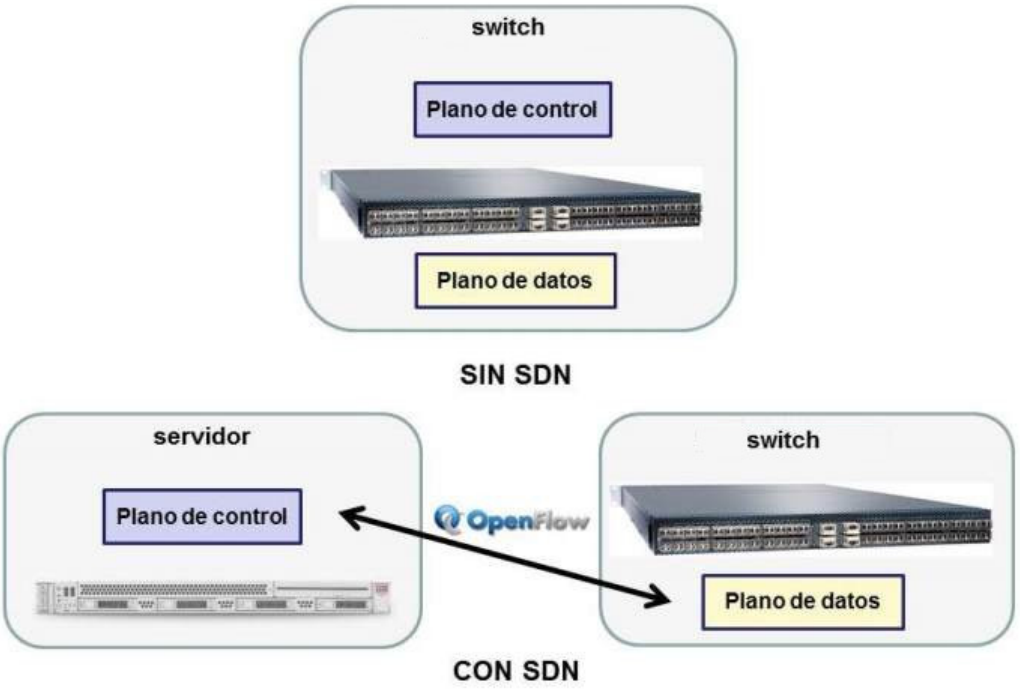

Figura 1.2: Nodos de redes convencionales y de redes definidas por software

1.4.2. Relación de las SDN con EPICSA

Con el transcurso de los años, cada vez son más las marcas del mundo de la informática que estudian, desarrollan y venden opciones SDN. La Diputación de Cádiz, estudia entonces incorporar a la red mencionada en el apartado anterior ciertos dispostivos SDN.

Se propone una reunión con el jefe del departamento de redes, la cual se recoge en el apartado "3.1.1 Documentación de partida"; y, como resultado de la misma, nace este proyecto. La finalidad de este es montar una pequeña topología en su CPD para determinar qué ventajas podría ofrecer el paradigma de las SDN a su red.

1.4.3. Dispositivos disponibles en EPICSA

Se muestra, a continuación, una relación de dispositos de EPICSA que están disponibles para su utilización para un posible laboratorio de pruebas:

- Dos conmutadores HP 2920 (Tabla 1.10)
- Un conmutador Mikrotik CRS125-24G-1S-IN (Tabla 1.11)

- Un blade Fujitsu BX600 S2 (Tabla 1.1)
- Un servidor HP ProLiant DL380 G5 (Tabla 1.8)

	Fujitsu Primergy BX600 S2 [12]
Tipo	Blade
Fuente de alimentación	4 x 2100W
Refrigeración	2 módulos
Servidores	Hasta 10
RAM	1 GB por servidor
Conmutadores	2 módulos de 6 entradas
Tamaño	7U
Precio	1.815,83

Tabla 1.1: Blade Fujistu BX600 S2

1.4.4. Situación actual en EPICSA

En la red de EPICSA, el alcance de la solución técnica relativa a la Transmisión de Datos e Internet, se desglosa en distintos apartados según el ámbito geográfico y funcional, como se muestra en la Figura 1.3, que ocupa dentro de la Red Corporativa de la Diputación Provincial de Cádiz.

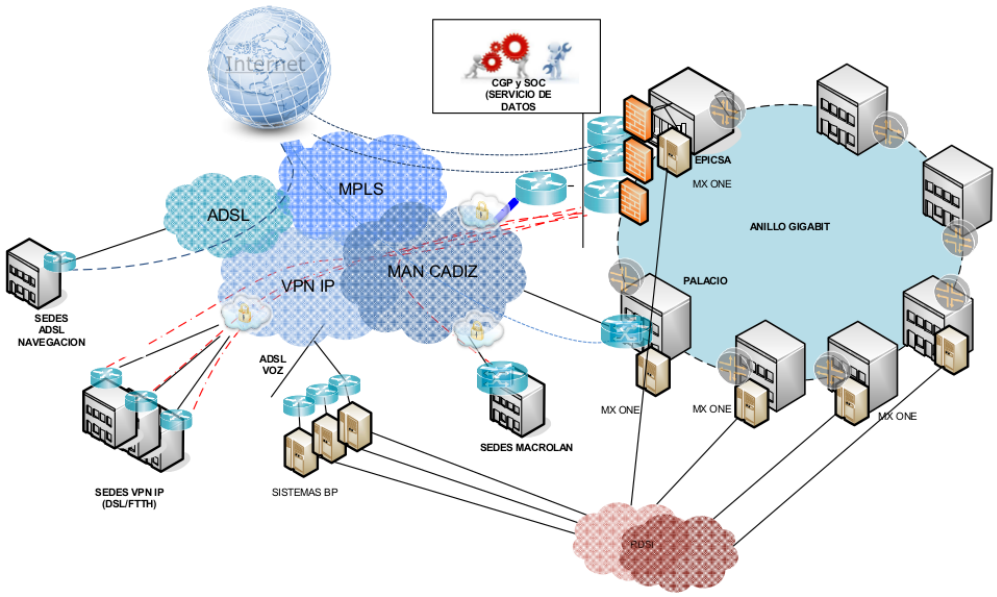

Figura 1.3: Arquitectura global de la red de EPICSA

1.4.4.1. Red de área metropolitana

Comprende la interconexión lógica en anillo de determinados emplazamientos en ámbito metropolitano ubicados en Cádiz capital; dispuestos como se presenta en la Figura 1.4. El domicilio de los distintos emplazamientos queda reflejado en la Tabla 1.2

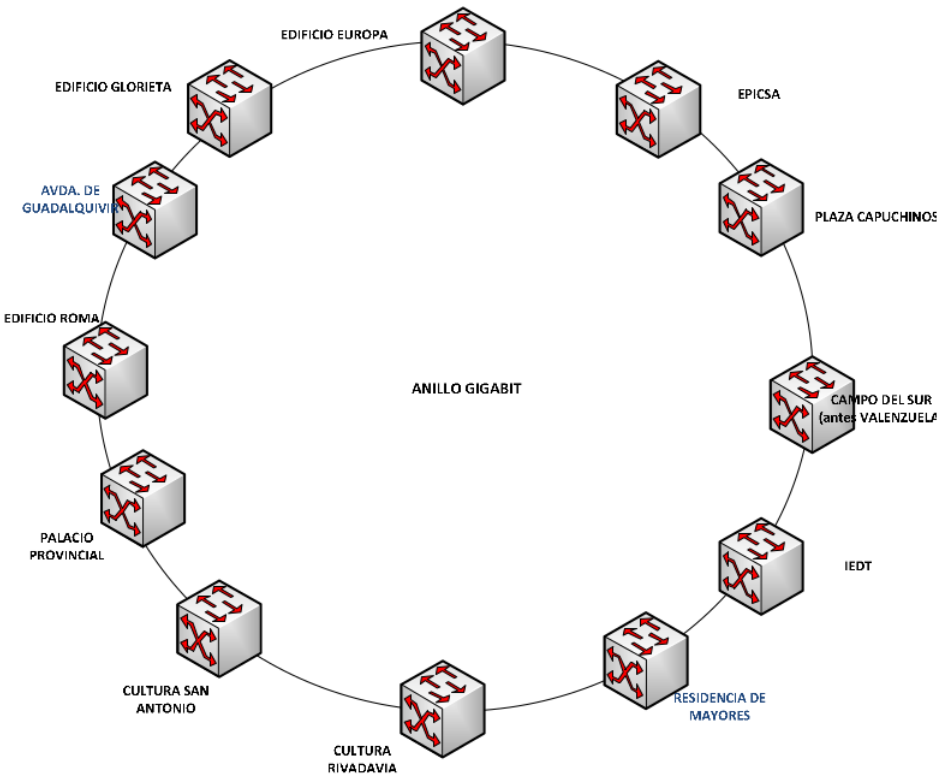

Figura 1.4: Topología de la red de área metropolitana

Sede	**Domicilio**
EPICSA	Edificio Carranza
Sede Central del SPD	Plaza de Capuchinos
Campo del Sur	Campo del Sur
Instituto de Empleo y Desarrollo Teconológico	Benito Pérez Galdós
Residencia de Mayores	Dr. Marañón
Fundación Provincial de Cultura	Edificio Rivadavia
Edificio San Antonio	Plaza San Antonio
Palacio Provincial	Plaza España
Edificio Roma	Avenida Ramón de Carranza
C.T.A. de Cádiz	Avenida Guadalquivir
Edificio Glorieta	Plaza Zona Franca
SPRyGT	Edificio Europa

Tabla 1.2: Emplazamientos de la red de área metropolitana

1.4.4.2. Red privada virtual

La RPV comprende la solución técnica necesaria para establecer la conectividad de todas las sedes externas al anillo metropolitano, mediante tecnología DSL o fibra. Estas sedes se conectarán con la sede principal ubicada en EPICSA en una topología en anillo como se muestra en la Figura 1.5

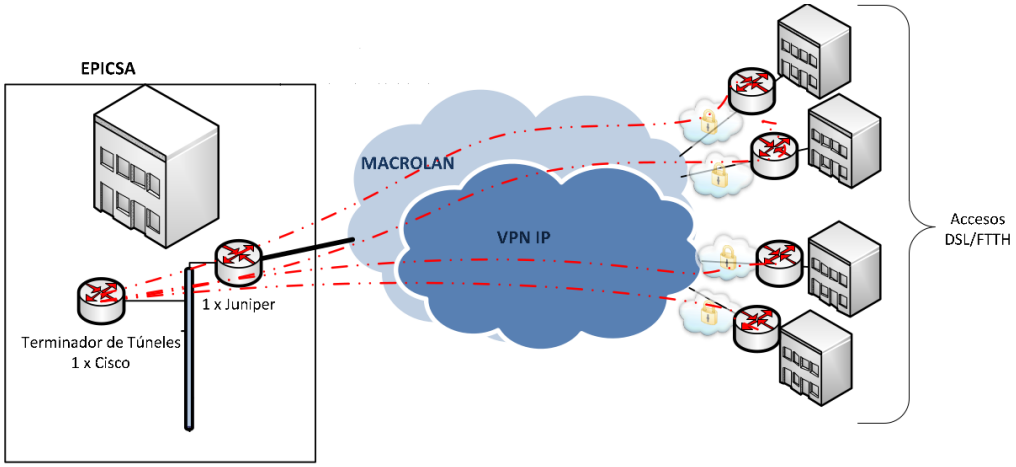

Figura 1.5: Topología de la red privada virtual

En las siguientes tablas se muestran todos aquellos centros que están dentro del alcance por las redes Macrolan/VPN IP.

Servicio Provincial de Recaudación y Gestión Tributaria	ARGISA Mancomunidad del Campo de Gibraltar
	PIT - Valdelagrana
	PIT - El Juncal
	PIT - Río San Pedro
	SPR Alcalá de los Gazules
	SPR Alcalá del Valle
	SPR Algar
	SPR Algeciras
	SPR Inspección Algeciras
	SPR Algodonales
	SPR Arcos de la Frontera
	SPR Barbate
	SPR Benalup
	SPR Bornos
	SPR Chipiona
	SPR Conil de la Frontera
	SPR Espera
	SPR Jerez de la Frontera
	SPR Jimena de la Frontera
	SPR Los Barrios
	SPR La Línea de la Concepción
	SPR Medina Sidonia
	SPR Olvera
	SPR Paterna de Rivera
	SPR Prado del Rey
	SPR Puerto Real
	SPR Puerto Serrano
	SPR El Puerto de Santa María
	SPR Rota
	SPR San Fernando
	SPR San José del Valle
	SPR Sanlúcar de Barrameda
	SPR San Martín del Tesorillo
	SPR San Roque
	SPR Tarifa
	SPR Trebujena
	SPR Ubrique
	SPR Unidad Técnica de Sanciones
	SPR Vejer de la Frontera
	SPR Villamartín

Tabla 1.3: Centros en la red privada virtual parte I

Ayuntamientos	Alcalá de los Gazules
	Alcalá del Valle
	Algar
	Algeciras
	Algodonales
	Barbate
	Benalup - Casas Viejas
	Benamahoma
	Benaocaz
	Bornos
	Castellar de la Frontera
	Chiclana de la Frontera
	Chipiona
	Conil de la Frontera
	El Bosque
	El Gastor
	El Puerto de Santa María - Serecop
	Espera
	Grazalema
	Jerez de la Frontera - JESSYTEL
	Jimena de la Frontera
	E.L.A. La Barca de la Florida
	La Línea de la Concepción
	Los Barrios
	Medina Sidonia
	Olvera
	Paterna de Rivera
	Prado del Rey
	Puerto Serrano
	Rota
	San José del Valle
	Sanlúcar de Barrameda
	San Fernando
	E.L.A. San Martín del Tesorillo
	San Roque
	Setenil de las Bodegas
	Tarifa
	Torre Alháquime
	Trebujena
	Ubrique
	Vejer de la Frontera
	Villaluenga del Rosario
	Villamartín
	Zahara de la Sierra
	E.L.A. Zahara de los Atunes

Tabla 1.4: Centros en la red privada virtual parte II

Oficinas del Servicio de Drogodependencias	CTA Algeciras
	Equipo de Apoyo en II.PP. Algeciras-Botafuegos
	CTA Chiclana
	CTA Alcalá de los Gazules
	CTA Barbate
	CTA Conil de la Frontera
	CTA Jerez de la Frontera
	Equipo de Apoyo en II.PP. Puerto I y Puerto II
	CTA La Línea de la Concepción
	CTA Puerto Real
	CTA San Fernando
	CTA Sanlúcar de Barrameda
	CTA Algodonales
	CTA Ubrique
	CTA Arcos de la Frontera
	CTA Villamartín
	CTA El Puerto de Santa María
	CTA Rota

Tabla 1.5: Centros en la red privada virtual parte III

Otros Centros	Fundación Medio Ambiente, Energía y Sostenibilidad de la Provincia de Cádiz
	Consorcio Bahía de Cádiz
	SAM Jimena de la Frontera
	SAM Medina Sidonia
	SAM Olvera
	Residencia Provincial de El Puerto de Santa María
	Residencia Provincial de la Línea de la Concepción
	Centro de Educación Ambiental El Castillejo
	Centro Agrícola Ganadero
	IFECA
	Parque Móvil

Tabla 1.6: Centros en la red privada virtual parte IV

1.5. Normas y referencias

1.5.1. Disposiciones legales y normas aplicadas

En esta sección se presenta el conjunto de disposiciones legales y las normas que se han tenido en cuenta en la elaboración y ejecución del proyecto.

- Norma UNE 157801:2007. Criterios generales para la elaboración de proyectos de sistemas de información.
- Real Decreto 994/1999, de 11 de Junio, por el que se aprueba el Reglamento de medidas de Seguridad de los Ficheros Automatizados que contengan Datos de Carácter Personal.

1.5.2. Bibliografía

[1] Carlos Manuel Rodríguez Alejandro García Susana Ballester. "Monografía de SDN". En: *Redes Definidas por Software*. 2015.

[2] Jose L. Muñoz Carlos Fernández. "Software Defined Networking (SDN) with OpenFlow 1.3, Open vSwitch and Ryu". En: *UPC Telematics Department*. 2015.

[3] Cisco. "Tema SDN CCNA RS 6.0". En:

[4] *Cisco Spine and Leaf Architecture Discussion - Nexus 5500 vs 6001*. en-US. Oct. de 2013. URL: http://thenetworksurgeon.com/cisco-spine-and-leaf-architecture-discussion-nexus-5500-vs-6001/ (visitado 22-06-2018).

[5] *diapositiva17.jpg (Imagen JPEG, 706 Œ 467 píxeles)*. URL: http://www.monografias.com/trabajos107/prototipos-redes-definidas-software/Diapositiva17.png (visitado 03-06-2018).

[6] *Diputación de Cádiz. EPICSA*. URL: http://www.dipucadiz.es/epicsa/ (visitado 04-06-2018).

[7] *Editing OpenDaylight OpenFlow Plugin:End to End Flows:Example Flows - OpenDaylight Project*. URL: https://wiki.opendaylight.org/view/Editing_OpenDaylight_OpenFlow_Plugin:End_to_End_Flows:Example_Flows (visitado 29-05-2018).

[8] Zen Garden. *Historia del SDN*. es-ES. Feb. de 2015. URL: https://openzen.wordpress.com/2015/02/12/historia-del-sdn/ (visitado 29-05-2018).

[9] *getfile.php.jpeg (Imagen JPEG, 1000 Œ 753 píxeles)*. URL: http://techpedia.fel.cvut.cz/html/getfile.php?tr=1500208049&oid=98&pid=1010&fid=MDVfZmxvd2NoYXJ0LmpwZw (visitado 03-06-2018).

[10] *Getting Started Ryu 4.24 documentation*. URL: http://ryu.readthedocs.io/en/latest/getting_started.html (visitado 29-05-2018).

[11] *Home*. en-US. URL: https://www.opendaylight.org/ (visitado 29-05-2018).

[12] *HP 2920-24G Switch - Conmutador - Gestionado - 24 x 10/100/1000 + 4 x Gigabit SFP compartido*. es. URL: https://www.mercadoactual.es/redes/switches-hubs/hp-2920-24g-switch-j9726a-abb-498811.html (visitado 06-06-2018).

[13] *HP 2920-24G Switch - Conmutador - Gestionado - 24 x 10/100/1000 + 4 x Gigabit SFP compartido*. es. URL: https://www.mercadoactual.es/redes/switches-hubs/hp-2920-24g-switch-j9726a-abb-498811.html (visitado 06-06-2018).

[14] *HP ProLiant DL380 G5 Specs*. en. URL: https://www.cnet.com/products/hp-proliant-dl380-g5-xeon-e5440-2-83-ghz-monitor-none-series/specs/ (visitado 07-06-2018).

[15] *La Universidad de Stanford lanza un programa para diseñar la Internet del futuro*. es. Mar. de 2007. URL: http://www.networkworld.es/actualidad/la-universidad-de-stanford-lanza-un-programa-para-disenar-la-internet-del-futuro (visitado 29-05-2018).

[16] *MikroTik*. en. URL: https://mikrotik.com/ (visitado 06-06-2018).

[17] *Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet*. URL: http://mininet.org/ (visitado 22-06-2018).

[18] *Narmox Spear*. URL: http://tutorial.spear.narmox.com/examples/#!/example1 (visitado 29-05-2018).

[19] *Narmox Spear - HP VAN SDN Controller*. URL: http://www.spear.narmox.com/ (visitado 29-05-2018).

[20] *Network Requirements Solutions (Part 2)*. es-ES. Mar. de 2017. URL: http://blogs.salleurl.edu/data-center-solutions/2017/03/network-requirements-solutions-part-2-a2017/ (visitado 22-06-2018).

[21] *Open Network Operating System*. en-US. URL: https://onosproject.org (visitado 29-05-2018).

[22] *OpenFlow, democratizando SDN ůů Entre Dev y Ops*. es-es. URL: https://www.entredevyops.es/posts/openflow-sdn.html (visitado 03-06-2018).

[23] *procesosoftware - Modelo Prototipado*. URL: https://procesosoftware.wikispaces.com/Modelo+Prototipado (visitado 21-06-2018).

[24] *Ryu Framework*. URL: https://osrg.github.io/ryu/ (visitado 29-05-2018).

[25] *sdndesacoplamientoplanoscontrolplanodatos.jpg (Imagen JPEG, 688 Œ 472 píxeles)*. URL: http://www.ramonmillan.com/imagenes/fotostutoriales/sdndesacoplamientoplanoscontrolplanodatos.jpg (visitado 29-05-2018).

[26] *Software-Defined Networking (SDN)*. URL: http://www.software-defined.net/networking.php (visitado 17-06-2018).

[27] *Web framework rankings | HotFrameworks*. URL: https://hotframeworks.com/languages/python (visitado 22-06-2018).

[28] *What are SDN Northbound APIs (and SDN Rest APIs)?* en-US. URL: https://www.sdxcentral.com/sdn/definitions/north-bound-interfaces-api/ (visitado 03-06-2018).

[29] *What are SDN Southbound APIs? - Where they are used.* en-US. URL: https://www.sdxcentral.com/sdn/definitions/southbound-interface-api/ (visitado 03-06-2018).

[30] *What is Cisco onePK?* en-US. URL: https://www.sdxcentral.com/cisco/datacenter/definitions/cisco-onepk/ (visitado 24-06-2018).

[31] *Wiki Home - ONOS - Wiki*. URL: https://wiki.onosproject.org/display/ONOS/ (visitado 29-05-2018).

1.5.3. Metodología seguida

En este proyecto se ha seguido el modelo de prototipos como metodología a emplear, es decir, se ha seguido un modelo de desarrollo evolutivo. Un modelo de prototipos tiene el siguiente ciclo de vida del `software`: [23]

1. **Recolección de requisitos**: Se definen aquellas características que definen al prototipo junto con el cliente.
2. **Diseño rápido**: Se hace un breve análisis sobre su composición.
3. **Construcción del prototipo**: Etapa de desarrollo.
4. **Evaluación del cliente**: Se le da al usuario una vista preliminar del prototipo resultante.
5. **Refinamiento del prototipo**: Si el cliente propone una mejora, se implementa.
6. **Producto de ingeniería**: Si el cliente considera que no es correcto o no cumple las expectativas, se desecha y se empieza la iteración de nuevo.

Figura 1.6: Ciclo de vida del modelo por prototipos

En nuestro caso, se ha seguido el mismo ciclo para cada uno de los siguientes prototipos, hasta conseguir la aplicación final que el cliente demandaba:

Figura 1.7: Prototipos de la aplicación

1.5.4. Programas utilizados

En este apartado se recoge una relación de las herramientas utilizadas en el desarrollo del proyecto o en las estimaciones del mismo:

- **GanttPro**: Generador de diagramas de Gantt para la visualización de la planificación temporal.
- **Draw.io**: `Software` para la construcción de diagramas de flujo, de proceso, topologías de red...
- **Papeeria**: `Software` para la redacción de documentos LaTeX.
- **GitHub**: `Software` para el control de repositorios.

- **Mininet**: `Software` que crea una red virtual realista, ejecutando `kernel`, conmutador y código de aplicación en una sola máquina. [17]

1.6. Definiciones y abreviaturas

1.6.1. Definiciones

- **Blade**: Tipo de computador simplificado con un diseño modular optimizado para minimizar el uso de espacio físico así como de energía. Normalmente se utilizan en los CPDs y alberga un conjunto de servidores físicos, o cuchillas, dentro de él.
- **Conmutador**: Dispositivo lógico que conecta otros dispostivos en una red y que opera en la capa de enlace de datos y, algunos, también en la capa de red del modelo OSI.
- **Framework**: Entorno de trabajo para el desarrollo e implementación de una aplicación.
- **Hardware**: Soporte físico.
- **Host**: Equipo conectado a una red.
- **Hub**: Dispositivo que actúa como concentrador para centralizar el cableado de una red. Opera en la capa física del modelo OSI.
- **Internet**: Sistema global de intercomunicación que utilizan el protocolo TCP/IP para la vinculación de dispositivos en todo el mundo.
- **Paquete**: PDU a nivel de capa de red en el modelo OSI. Está compuesta por una cabecera, un área de datos y una cola.
- **Router**: Dispositivo de interconexión de redes que trabaja en la capa de red del modelo OSI. Su función es la de encaminar los paquetes de datos de una red a otra.
- **Software**: Conjunto organizado de instrucciones, datos, reglas e interrelaciones, que cuando son cargados en el área de ejecución de programas del ordenador, permite operar al ordenador.

1.6.2. Abreviaturas

Se presenta, a continuación, en la Tabla 1.7 un listado de abreviaturas que se han utilizado durante la elaboración de este documento, y sus respectivos significados. A continuación de la misma se muestra una serie de de definiciones para la comprensión de este proyecto.

Abreviatura	**Definición**
API	Interfaz de Programación de Aplicaciones
CPD	Centro de Procesamiento de Datos
CPU	Unidad Central de Procesamiento
EPL	Licencia Pública de Eclipse
HTTP	Protocolo de Transferencia de Hipertexto
ICMP	Protocolo de Control de Mensajes de Internet
IETF	Grupo de Trabajo de Ingeniería de Internet
IP	Protocolo de Internet
MAC	Control de Acceso al Medio
NAT	Traducción de Direcciones de Red
ONF	Fundación de Open Networking
PDU	Unidad de Datos de Protocolo
SDN	Redes Definidas por Software
SGBD	Sistema de Gestión de Base de Datos
SSH	Intéprete de Órdenes Seguro
STP	Protocolo de Árbol de Expansión
TCAM	Tabla de Memoria de Contenido Direccionable
URL	Localizador Uniforme de Recursos
UTF	Formato de Transformación Unicode
VAN	Red de Aplicaciones Virtuales
VPN	Red Privada Virtual

Tabla 1.7: Abreviaturas

1.7. Requisitos iniciales

1. Reducir la complejidad de la estructura de la red.
2. Conseguir actualizaciones en la red sin impactos adversos.
3. Ofrecer un control centralizado de la estructura de red.
4. Hacer un sistema confiable y seguro.
5. Automatizar tareas.
6. Ofrecer una interfaz visualmente cómoda de gestión.
7. Desarrollar un sistema de políticas de rechazo de paquetes.
8. Implementar rutas de respaldo para servidores.

1.8. Alcance

Este proyecto se aplica al estudio de diferentes controladores SDN y al diseño e implementación de un prototipo de aplicación SDN en la red de la Diputación Provincial de Cádiz. Durante su ejecución se ha llevado a cabo las siguientes tareas:

- Estudio teórico de la red SDN y de cada uno de los componentes que la conforman.
- Manuales de instalación, configuración y uso de diferentes controladores así como de otras aplicaciones a emplear.
- Estudio de mercado de los controladores con objeto de determinar el más adecuado para el proyecto.
- Desarrollo de diferentes prototipos utilizando el controlador elegido, sobre una red simulada como paso previo a la implementación sobre conmutadores reales de EPICSA.
- Desarrollo de una aplicación SDN que actuará sobre los conmutadores reales de EPICSA por medio del controlador elegido.

1.9. Estudio de alternativas y viabilidad

Para la implementación de la aplicación SDN se han tenido en cuenta los distintos niveles de la arquitectura, y las distintas alternativas para cada uno.

Lo primero que tenemos que evaluar es que los conmutadores que trabajan con protocolos de SDN funcionarán de una manera distinta a los dispositivos tradicionales (Figura 1.8 [26]) y, por ello, debemos valorar que el diseño de la red puede ser diferente.

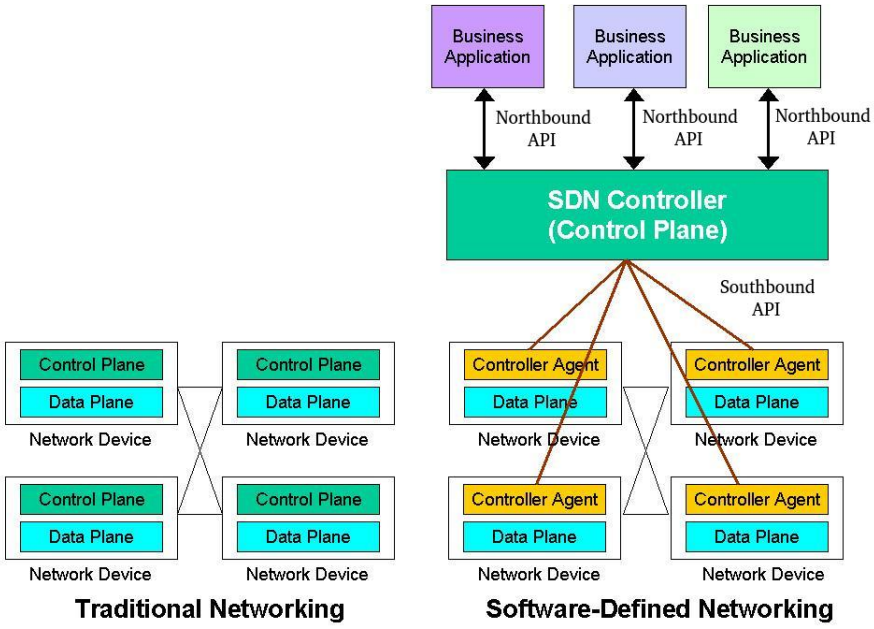

Figura 1.8: Alternativa a los nodos tradicionales

En esta sección estudiaremos distintas alternativas `Southbound APIs`, responsables de la comunicación del controlador con los dispositivos de red, y de controladores SDN, encargados de suministrar la información necesaria a dichos dispostivos; y estimaremos cuál puede ser la mejor opción de entre todas ellas.

Una vez elegido el controlador SDN, estudiaremos qué `framework` utilizar para el plano de aplicación.

1.9.1. Southbound APIs

Existe un número limitado de `Southbound APIs` actualmente. Esto es debido a dos posibles razones:

1. Son APIs propietarias de una marca en concreto, como por ejemplo **OnePK** de Cisco, la cuál la hace recomendable si estamos trabajando con equipos de la misma marca, pero no si necesitamos usar otros conmutadores.

2. Nacieron como protocolos de gestión y algunos investigadores quisieron convertirlos en APIs, como por ejemplo **XMPP**, **NETCONF** o **SNMP**. Su problema es que fueron desarrolladas desde el principio para servir a propósitos específicos y su aplicabilidad queda muy limitada.

Por ello, la `Southbound API` más famosa hoy en día es **OpenFlow**: es de código abierto, compatible con múltiples marcas y funciona correctamente como API de propósito general de SDN; y la convierte en la mejor alternativa.

1.9.1.1. Modelo jerárquico y modelo "Spine and Leaf"

Debemos mencionar dos modelos de red típicamente usados con conmutadores tradicionales:

- **Modelo jerárquico** [20]: Está compuesto por tres capas:
  - Capa de Acceso: Conexión con los dispositivos finales.
  - Capa de Distribución: Es la capa intermedia entre la capa de acceso y la capa de núcleo y hace posible la conectividad entre ambas.
  - Capa de Núcleo: Se encarga de transportar datos entre las diferentes redes.
- **Modelo `Spine and Leaf`** [4]: Está compuesto por dos tipos de conmutadores:
  - Conmutadores `Leaf`: Son los conmutadores que están conectados con los dispositivos finales, y están conectados, cada uno de ellos, con todos los conmutadores `spine`.
  - Conmutadores `Spine`: Estos conmutadores poseen una alta densidad de puertos por temas de escalabilidad y están directamente conectados con el dispositivo habilitado para enrutar paquetes fuera de la red. Además, no existe interconexión entre los conmutadores `Spine`.

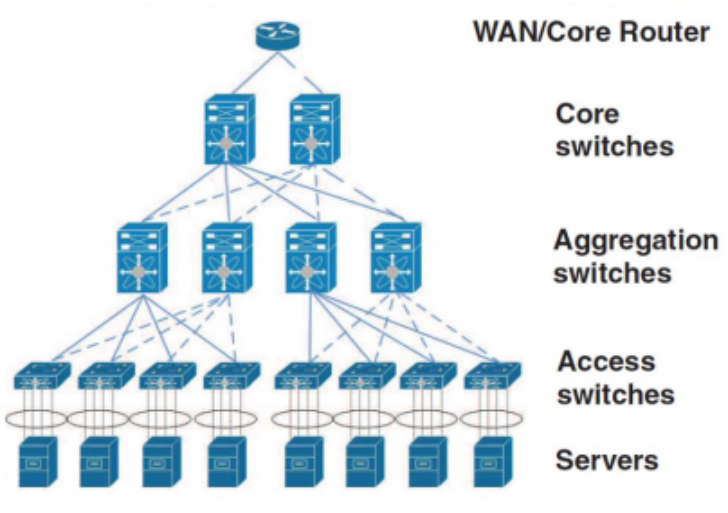

Figura 1.9: Modelo jerárquico

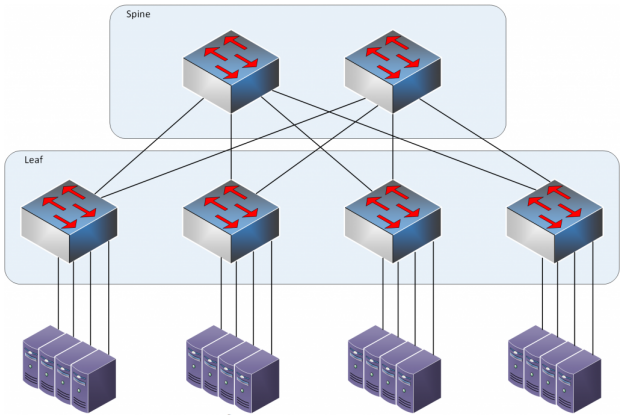

Figura 1.10: Modelo Spine and Leaf

En una arquitectura SDN, los conmutadores `Spine` estarían directamente conectados con el controlador; y en el modelo jerárquico serían los dispositivos de la capa de núcleo los que dependerían del plano de control.

Idílicamente, el modelo `Spine and Leaf` posee más ventajas que el modelo jerárquico, ya que provee redundancia y escalabilidad; pero tiene la desventaja del coste económico. Puesto que EPICSA no ha implementado un modelo `Spine and Leaf`, los prototipos que se desarrollan en los apartados siguientes tampoco seguirán ese modelo.

1.9.2. Simulación de dispositivos OpenFlow

Para una correcta valoración, se ha usado el `software` Mininet, el cuál permite simular una topología SDN a partir de que se le especifique una dirección IP para el controlador.

Mininet trabaja con conmutadores virtuales, pero actualmente son cada vez más las marcas que atienden la demanda de dispositivos que puedan trabajar con el protocolo OpenFlow. En la parte virtual, Mininet configura los conmutadores de modo transparente para el usuario ya que estos no son de ninguna marca concreta; y, por tanto, no guarda relación con los conmutadores del mundo real. Estos últimos tienen a disposición del usuario un manual de configuración para el protocolo OpenFlow, más o menos extenso, dependiendo de la marca.

Existen tres maneras de simular una topología en Mininet (véase Anexo 3.2 Mininet):

1. Mediante comandos.
2. Mediante el `script` “Miniedit”.

3. Mediante la implementación de archivos python.

Nos decantamos por la última opción a razón de que nos da la máxima libertad de personalización de una topología.

Para probar la mejor alternativa se hará uso de la misma topología en todas ellas, recogida en el archivo test.py, usando un ordenador simulando ser un servidor que haga la función de controlador.

Una alternativa sería disponer de un servidor real sobre el que correr Mininet o que directamente le colgaran tres conmutadores capaces de trabajar con OpenFlow; el resultado no variaría, son los mismos comandos y la misma interfaz de consola, lo único que cambiaría sería la necesidad de instalar SSH en el servidor para su configuración de manera remota.

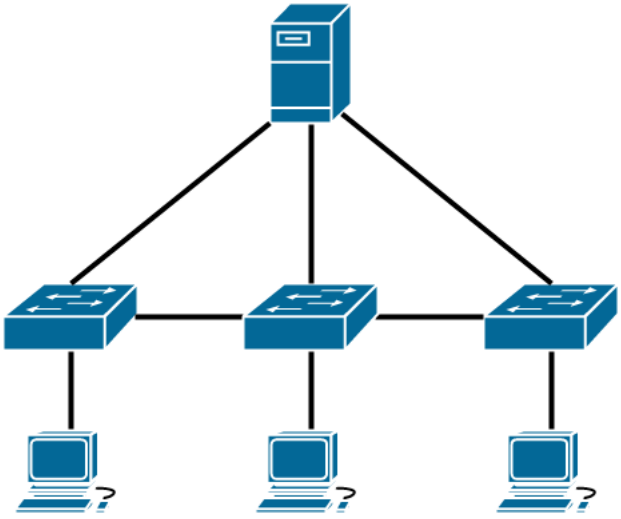

Figura 1.11: Topología de prueba test.py

Si se ha ejecutado correctamente, podemos usar el comando `net` para que muestre una descripción de la red, y obtendríamos una pantalla en Mininet como la de la Figura 1.12

```
rewtelize@ragnar:~/Escritorio$ sudo python test.py
Unable to contact the remote controller at 127.0.0.1:6633
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3
c0
mininet>
```

Figura 1.12: Pantalla de Mininet para test.py

1.9.3. Controladores SDN

Un controlador SDN toma el rol de "cerebro" de la topología, es decir, transmite información a los `switches` que le cuelgan (a través de la `Southbound API`) y a las aplicaciones y la lógica de negocio que están en el plano de aplicación (a través de la `Northbound API`).

En esta sección se estudiarán algunos de los `softwares` que están contando con más fama en el mercado, como son OpenDaylight, Ryu, HP VAN SDN u ONOS.

1.9.3.1. OpenDaylight

OpenDaylight es un proyecto mantenido por `The Linux Foundation`, desarrollado en Java, y su uso está disponible bajo Licencia Pública de Eclipse (EPL); por ello, cualquiera puede participar en su desarrollo. Opendaylight, a su vez, ha ayudado y es básico en multitud de proyectos SDN de código libre orientados a la producción de soluciones tecnológicas relevantes y confiables. [11]

Con la configuración básica que trae por defecto OpenDaylight nos encontramos con unos tiempos de latencias un poco altos, pero tiene la ventaja de que crear un flujo resulta más sencillo que otros controladores, ya que se pueden implementar mediante lenguaje XML y etiquetas predefinidas.

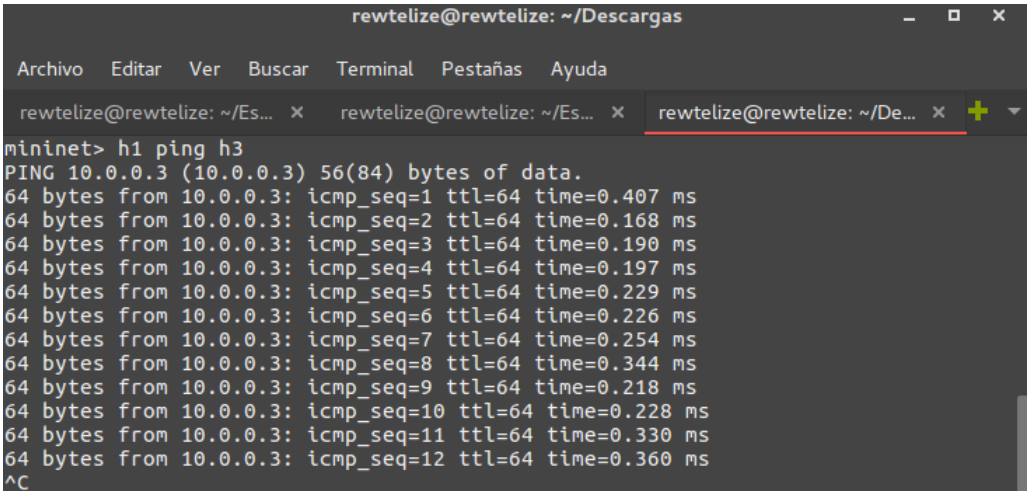

Figura 1.13: OpenDaylight en prueba test.py

1.9.3.2. Ryu

Ryu es un `framework` para SDN. Proporciona, a través de una API bien definida, componentes `software` que facilita a los desarrolladores la creación de nuevas aplicaciones SDN. [24]

Si probamos a cargar la topología test.py en Mininet e introducimos la configuración l2.py comprobamos que, con muchas menos líneas que el resto de controladores, conseguimos crear una topología SDN con conectividad entre todos los dispositivos.

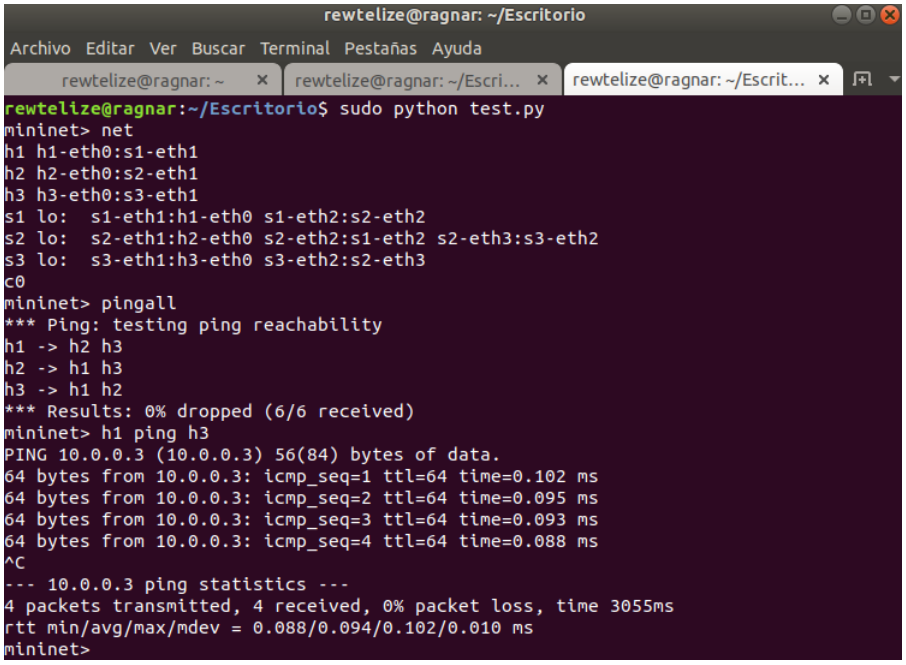

Figura 1.14: Ryu en prueba test.py

1.9.3.3. HP VAN SDN

El controlador de `Hewlett-Packard` (HP) proporciona un único punto de control unificado sobre una red SDN, simplificándo y orquestándola de una manera más cómoda para el administrador.

El controlador VAN SDN está diseñado para operar en entornos de campus, centro de datos o proveedor de servicios. [19]

HP VAN SDN posee una interfaz visualmente cómoda para realizar las operaciones y, utilizando su configuración ya implementada para el envío de paquetes, obtenemos los siguientes resutados:

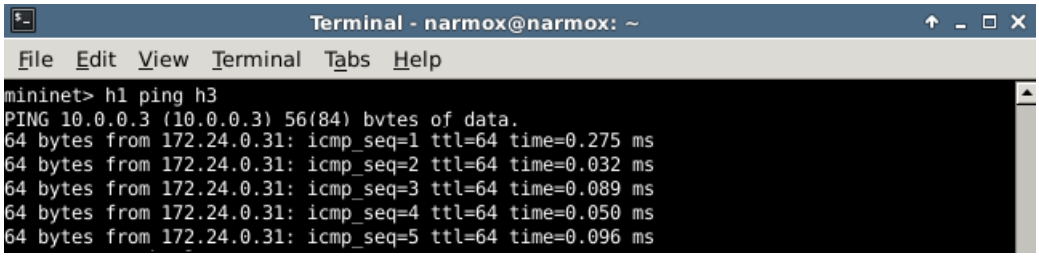

Figura 1.15: HP VAN SDN en prueba test.py

1.9.3.4. ONOS

ONOS se define como la única plataforma controladora SDN que admite la transición de redes `brownfield` (dispositivos nuevos que deben coexistir con la tecnología heredada) a redes SDN `greenfield` (dispositivos nuevos que no dependen de la tecnología heredada). Es de código abierto y proveen escalabilidad y gran redimiento entre muchos otros factores. [21]

Utilizando su configuración básica, de más de 900 líneas, hemos conseguido que todos los dipositivos esten conectados entre ellos; pero el hecho hipotético de resolver errores en un futuro puede volverse realmente tedioso y costoso en tiempo.

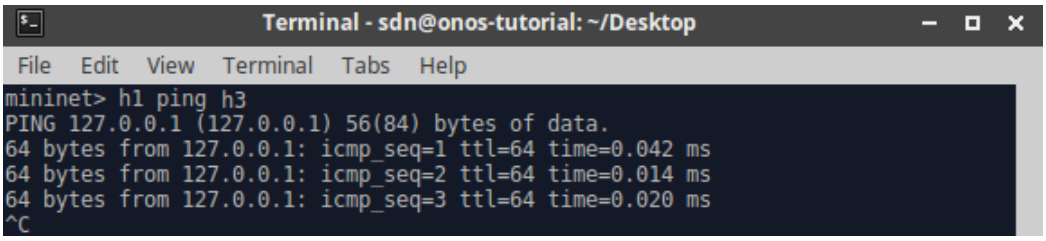

Figura 1.16: ONOS en prueba test.py

1.10. Descripción de la solución propuesta

Tal y como se ha especificado en el apartado 1.5.3 Metodología seguida, se sigue un modelo de prototipos, es decir, se han desarrollado versiones rápidas que el cliente ha evaluado y ha dado su visto bueno.

Este apartado recoge dichos prototipos y un estudio de los controladores y el porqué de la elección del más idóneo; de forma que quede descrita la propuesta realizada y las características que lo definen.

1.10.1. Elección del controlador

Para nuestro laboratorio de prácticas utilizaremos el siguiente servidor como controlador:

	HP ProLiant DL380 G5 [14]
Tipo	Servidor
CPU	Intel Xeon E5440 / 2.83 GHz
Memoria	32 GB
RAM	4 GB
Puertos Ethernet	2 x Gigabit Ethernet + Posible extensión
Sistemas operativos certificados	Microsoft Windows 2000 Server Microsoft Windows Server 2003 Novell NetWare Red Hat Linux, SCO OpenServer SuSe Linux, SunSoft Solaris 10 UnitedLinux, UnixWare
Tamaño	2U
Precio (€)	751.25

Tabla 1.8: Servidor HP ProLiant DL380 G5

En la Tabla 1.9 describimos las distintas características de los controladores utilizados con el fin de seleccionar uno de ellos para una implementación personalizada de la aplicación SDN.

	OpenDaylight	**Ryu**	**HP**	**ONOS**
Primera versión	Feb. 2014	Sept. 2013	Nov. 2013	Dic. 2014
OpenFlow	v1.0	v1.0 - v1.2 v1.3	v1.0 - v1.3	v1.0 - v1.3
Implementación	Java	Python	Java	Java
Flujos	XML	Python	Mediante Componentes	Java
REST API	Sí	Sí	Sí	Sí
Escalable	Sí	Sí	Dependencia con Componentes	Sí
Documentación	Buena	Buena	Media	Media
Comunidad	Media	Alta	Baja	Alta
Soporte	Linux, MAC, Windows	Linux, MAC, Windows	Linux, MAC, Windows	Linux, MAC, Windows
Código abierto	Sí	Sí	No	Sí
Precio (€)	-	-	2.110	-

Tabla 1.9: Comparativa de controladores

Entre las distintas opciones, podríamos valorar que usar el **controlador Ryu** es la

opción más acertada, debido a los siguientes motivos:

- Es código abierto, es decir, cualquiera puede colaborar en la mejora del código del controlador y compartirlo. Con **HP VAN SDN** no tenemos esa opción, el código se lo reserva la empresa HP y por tanto estamos sujetos a sus propias mejoras.
- Al ser libre, el precio dependerá en la medida que queramos colaborar altruístamente, pudiendo no pagar nada por la licencia de uso.
- La elaboración de flujos se hace con Python. Con este lenguaje tenemos multitud de opciones de desarrollo; desde poder elaborar distintos tipos de archivos hasta generar comunicaciones con bases de datos y servicios en la red. **OpenDaylight** utiliza XML, el cuál no presenta un abanico de opciones tan variado como Python.
- Tiene soporte para las versiones 1.0, 1.2 y 1.3 del protocolo OpenFlow, **OpenDaylight** solo presenta la versión 1.0; reduciéndo así el número de dispositivos compatibiles.
- En general, una misma aplicación desarrollada tanto en **ONOS** como en Ryu, tiene menos líneas de código en este último y hace que el código sea mucho más mantenible con el tiempo.

1.10.2. Elección del framework para el desarrollo de la aplicación SDN

Debido a que concluímos que Ryu es el controlador idóneo para esta aplicación, y este implementa las aplicaciones mediante lenguaje python, debemos buscar un `framework` de desarrollo para nuestra interfaz que utilice el mismo lenguaje. A continuación se muestra un ranking con los `frameworks` que trabajan en python, dicha encuesta fue realizada en la web `https://hotframeworks.com/languages/python` [27]

Framework	Score
Django	92
Flask	84
Tornado	73
Bottle	65
web.py	63
Pyramid	63
web2py	61
CherryPy	55
Grok	47
Zope	46
TurboGears	41
Tipfy	36
Quixote	0

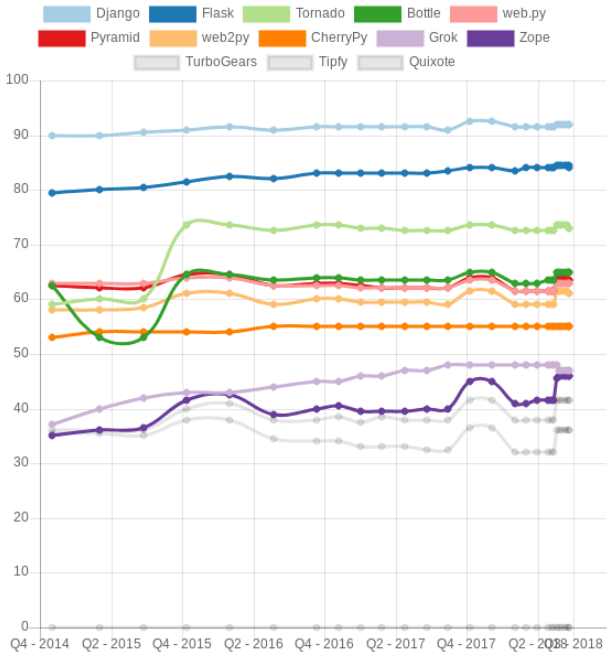

Figura 1.17: Ranking frameworks de desarrollo en python

De entre todos, valoraremos cuál de los dos primeros es la elección más idónea para la elaboractión de la interfaz de la aplicación SDN: Django o Flask.

Por un lado, Django cuenta con la comunidad más activa de los `frameworks` de desarrollo en python. Posee un registro de 80.000 preguntas en "StackOverflow" y existen multitud de blogs de desarrolladores y usuarios avanzados. La comunidad de Flask no es tan grande, tiene registradas sólo 5.000 preguntas en "StackOverflow", lo cuál la hace 15 veces más pequeña que Django.

Si echamos un vistazo a la web de repositorios libres, "GitHub", observamos que poseen un número casi idéntico de estrellas: 31.543 en el caso de Flask y 30.164 de Django. En cuanto a número de descargas (`forks`), "GitHub" tiene registrada 9.936 en el caso de Flask contra los 12.712 de Django.

Después de este estudio, decidimos escoger **Django** como el `framework` de desarrollo para nuestra interfaz web.

1.10.3. Elección de los conmutadores

En este caso utilizaremos los conmutadores, capaces de trabajar con el protocolo OpenFlow, que EPICSA tiene disponible actualmente: dos conmutadores HP 2920

(de ahora en adelante, HP-1 y HP-2) y un conmutador Mikrotik CRS125-24G-1S-IN.

	HP 2920 [13]
Tipo	Switch
Tecnología Ethernet	Gigabit Ethernet
Tasa de reenvío	10/100/1000Base-T
Puertos	24G + 4SFP
QoS	Sí
TCAM	16000 entradas
Tamaño	1U
Memoria Interna	512 MB
VLANs	Sí
Precio	1.114,59
OpenFlow	Sí

Tabla 1.10: Conmutador HP 2920

	Mikrotik CRS125-24G-1S-IN [16]
Tipo	Switch
Tecnología Ethernet	Gigabit Ethernet
Tasa de reenvío	10/100/1000Base-T
Puertos	24G +1SFP
QoS	Sí
Tamaño	1U
Memoria Interna	128 MB
VLANs	Sí
Precio	160,49
OpenFlow	Sí

Tabla 1.11: Conmutador Mikrotik CRS125-24G-1S-IN

1.10.4. Desarrollo de prototipos

En esta sección, se muestra a continuación, los hitos que se han implementado durante la metodología basada en el modelo por prototipos; empezando imbuyendo una primera aplicación Ryu en un conmutador y terminando con una web, visualmente cómoda para el administrador, capaz de gestionar dichas aplicaciones sobre varios conmutadores.

Prototipo	**Descripción**
1: Comportamiento de switch	El servidor controlador creará una tabla en la que almacene el conmutador junto con una dirección física y un puerto. De esta manera, cuando llegue un paquete, el controlador le dirá al conmutador qué hacer con él: si existe una entrada en su tabla, lo mandará por el puerto que tiene asociado; si no, hará una inundación.
2: Conexión de varios switches al mismo controlador	El conmutador, una vez que el controlador responde por qué puerto darle salida al paquete, generará un flujo para no tener que volver a preguntarle. El flujo contendrá las direcciones físicas origen y destino y el puerto de salida. Así, cuando llega un paquete, lo primero que comprobará es si tiene un flujo que sabe qué hacer con él; y, en caso contrario, preguntar al servidor controlador.
3: Interconexión de switches en forma de anillo	Este prototipo consigue simular la red del área metropolitana de EPICSA y, para ello, ha sido necesario implementar el protocolo STP en la aplicación Ryu. Es por esto, que el servidor controlador ha decidido qué enlace bloquear y ha dado respuesta a las peticiones de los conmutadores en consecuencia a ello.
4: Implementación de políticas	Una política se define como una relación entre un puerto origen y/o un puerto destino. Si el administrador de la red, crea una política, entonces el servidor controlador no dejará que los conmutadores creen un flujo con ese puerto origen y/o con ese puerto destino.
5: Implementación rutas de respaldo	Dados dos servidores (siendo el segundo un servidor de `backup` del primero), el servidor controlador pedirá a ambos sus contenidos. Si el primer servidor no está disponible, entonces el controlador mostrará el contenido del segundo servidor. Si estamos en el segundo caso, el servidor controlador mandará un email al administrador de red alertando de esa incidencia.
6: Aplicación web SDN	Para conseguir una interfaz cómoda para que el administrador de la red pueda, dentro del propio servidor controlador, gestionar diferentes aplicaciones ryu, visualizar la topología de red que está actualmente activa, mostrar gráficas de rendimiento de los conmutadores... se ha creado una aplicación web que lo ayude a gestionar una arquitectura SDN.

Tabla 1.12: Prototipos y descripciones

1.10.4.1. Prototipo 1: Comportamiento de switch

1.10.4.1.1 Objetivo

Una vez instalada e implementada la primera aplicación de Ryu en un conmutador (véase Anexo 3.5), el primer paso debe ser implementar una tabla de direcciones MACs para que el dispositivo no se vea obligado a realizar una inundación cada vez que le llegue un paquete: lo cuál lo hace más seguro y reduce latencias.

1.10.4.1.2 Implementación

En esta implementación creamos una tabla en python (mac_to_port) que sustituya a la tabla de direcciones MACs que dispondría un conmutador, y vamos asociando a cada MAC un puerto según el tráfico. Si tenemos la MAC destino (dst) registrada en la tabla, lo mandamos por el puerto asociado; si no, hacemos inundación (OFPP_FLOOD).

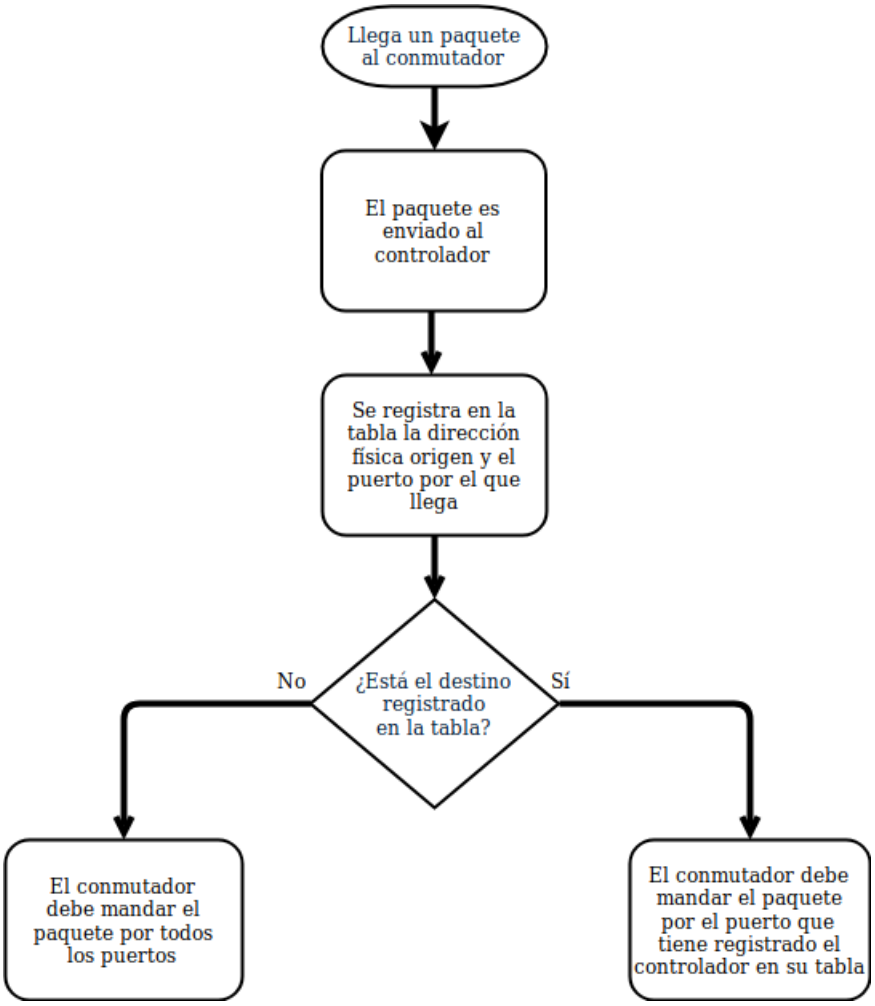

Figura 1.18: Diagrama de flujo del Prototipo 1

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_0
from ryu.lib.mac import haddr_to_bin
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
import sys

from ryu.lib.packet import packet, ethernet, arp, ipv4
import array
```

```
class SimpleSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]
    print("[DEDALO] Inicio aplicacion SimpleSwitch")

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    # Llega un paquete procedente del conmutador
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # Se desglosa el paquete en mensaje (msg), camino de datos
        # seguido (datapath) y protocolo OpenFlow tratado (ofproto)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto

        self.logger.info("[DEDALO] Direccion: " +
            str(datapath.address))
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)

        # Obtencion de las direcciones fisicas origen y destino
        dst = eth.dst
        src = eth.src

        # La variable dpid contiene el identificador unico del
        # conmutador
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        # Se registra en la tabla la direccion fisica origen y
        # el puerto por el que llega
        self.mac_to_port[dpid][src] = msg.in_port

        # Esta el destino registrado en la tabla?
        # Si: El conmutador debe mandar el paquete por el puerto
        # out_port
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        # No: El conmutador debe mandar el paquete por todos los
        # puertos
```

```
        else:
            out_port = ofproto.OFPP_FLOOD

        actions =
            [datapath.ofproto_parser.OFPActionOutput(out_port)]

        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        print("[DEDALO] Paquete entrante en dispositivo " +
            str(dpid2) + " mac origen " + str(src2) +
            " mac destino " + str(dst2))
        print("[DEDALO] Puerto de entrada " +
            str(msg.in_port) + " y puerto de salida " +
            str(out_port))

        # Datos del paquete de salida
        out = datapath.ofproto_parser.OFPPacketOut(
            datapath=datapath,
            buffer_id=msg.buffer_id,
            in_port=msg.in_port,
            actions=actions, data=data)

        print("[DEDALO] Tabla CAM: " + str(self.mac_to_port) + "\n")

        # El paquete es enviado al conmutador
        datapath.send_msg(out)

    # Si se conecta o desconecta un equipo al conmutador, este lo
    # comunica al controlador para que lo saque o lo inserte
    # en su tabla
    @set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
    def _port_status_handler(self, ev):
        msg = ev.msg
        reason = msg.reason
        port_no = msg.desc.port_no

        ofproto = msg.datapath.ofproto
        if reason == ofproto.OFPPR_ADD:
            self.logger.info("puerto insertado %s", port_no)
        elif reason == ofproto.OFPPR_DELETE:
            self.logger.info("puerto eliminado %s", port_no)
```

```
        elif reason == ofproto.OFPPR_MODIFY:
            self.logger.info("puerto modificado %s", port_no)
        else:
            self.logger.info("Estado desconocido %s %s", port_no,
                             reason)
```

1.10.4.1.3 Simulación

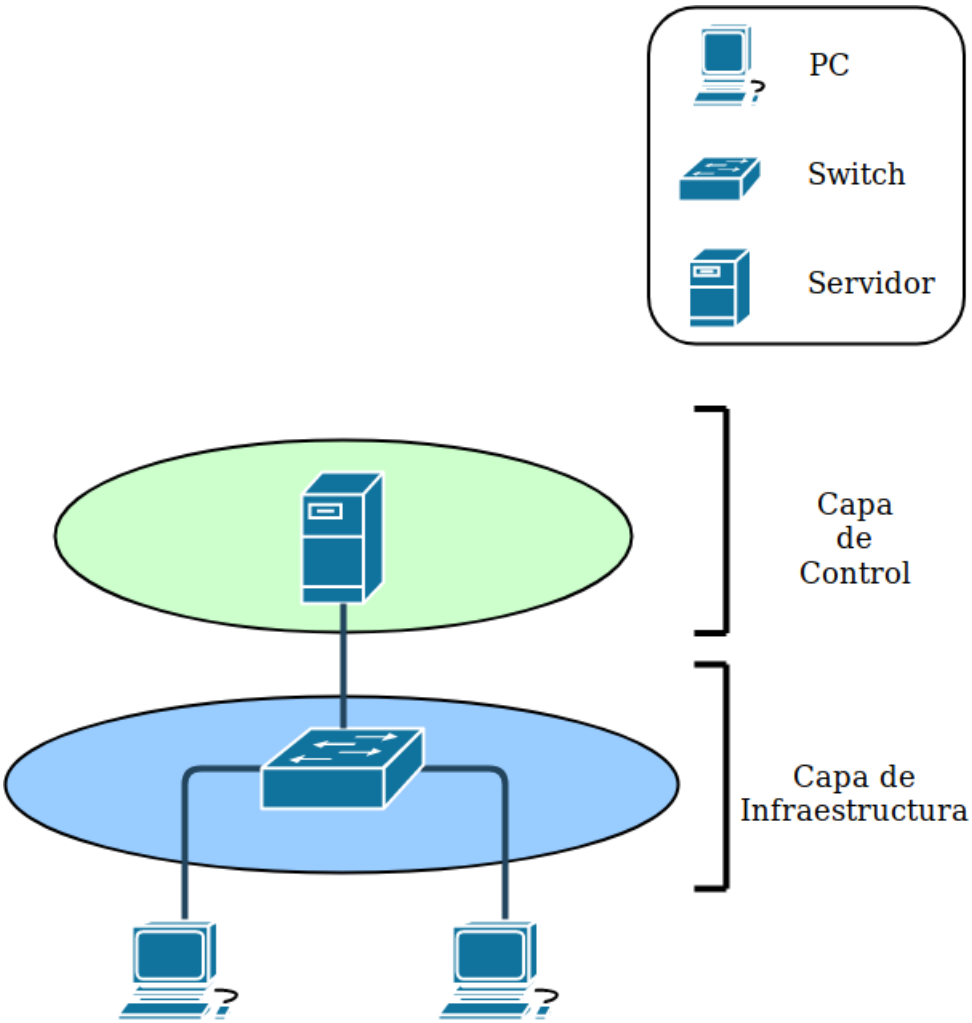

Figura 1.19: Topología simulada del Prototipo 1

Para simular esta topología sobre nuestro equipo, el cuál hará de servidor controlador, pondremos el siguiente comando en la consola:

$ sudo mn –controler=remote,ip=127.0.0.1 –topo single,2 –mac –switch ovsk,protocols=OpenFlow10

Nota: Para más información sobre los comandos empleados, véase Anexo 3.2 Mininet.

Una vez lanzados Mininet y Ryu a ejecución en dos consolas diferentes (para más información véase Anexo 3.2 Mininet y Anexo 3.5 Ryu respectivamente), obtendremos las siguientes pantallas. La primera mostrará que ambos `hosts` tienen conectividad, confirmando que la aplicación está funcionando; la segunda es la de Ryu y mostrará los mensajes de las funciones `print` que se han puesto en el código de la aplicación, tal y como se comenta en el mismo, los tres mensajes se mostrarán por cada paquete que llegue al controlador.

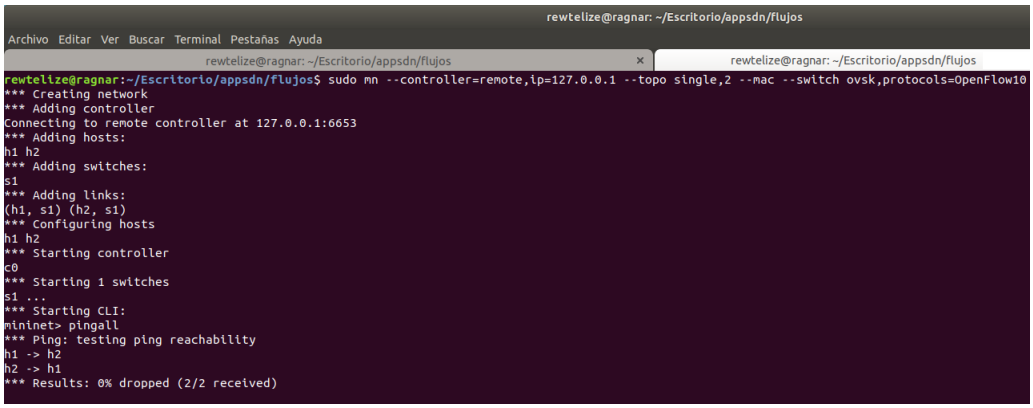

Figura 1.20: Simulación del Prototipo 1 - Parte I

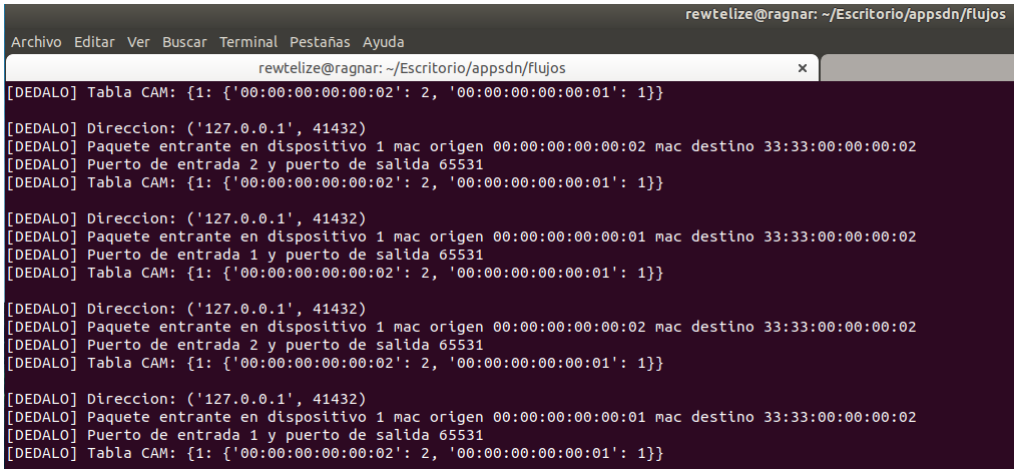

Figura 1.21: Simulación del Prototipo 1 - Parte II

1.10.4.1.4 Entorno real

En el CPD de EPICSA, se crea, físicamente, la topología:

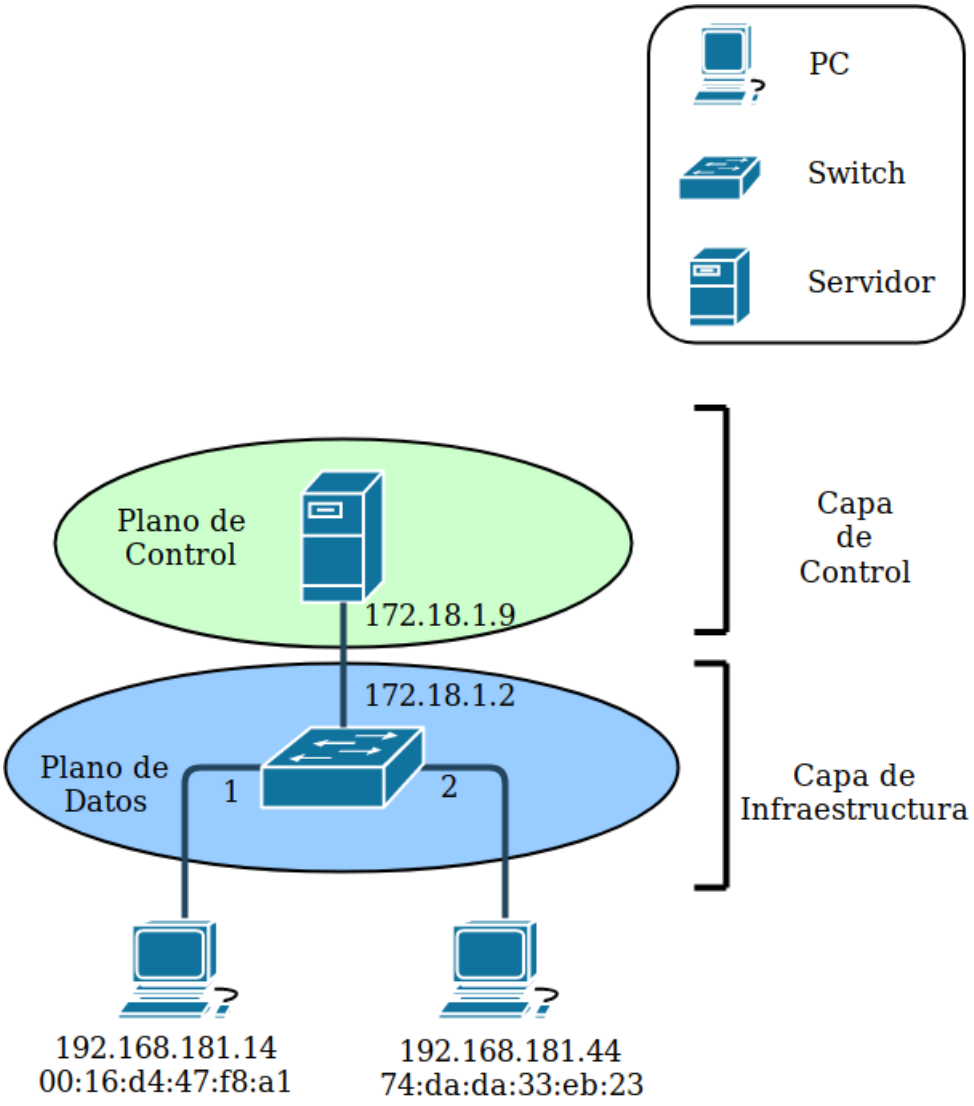

Figura 1.22: Topología real del Prototipo 1

- El servidor HP ProLiant tiene el rol de controlador.
- Un `switch` HP 2920 será el conmutador conectado al controlador (vlan 50).
- Dos ordenadores disponibles completarán la arquitectura: 192.168.181.14 y 192.168.181.44 (vlan 3).

Una vez configurados, tendremos que preparar al conmutador para que trabaje

con el protocolo OpenFlow, para ello habilitamos el protocolo y creamos una instancia con la que indicar al conmutador cómo encontrar al controlador y sobre qué redes trabajar.

S1(config)# openflow controller-id 1 ip 172.18.1.9 controller-interface vlan 50
S1(config)# openflow instance aggregate controller-id 1
S1(config)# openflow instance aggregate enable
S1(config)# openflow enable

Nota: Para más información sobre los comandos empleados, véase Anexo 3.3. Configuración de los connmutadores.

Finalmente lanzamos la aplicación ryu en la consola del servidor y comprobamos que existe conectividad entre ambos `hosts` y que la tabla del controlador contiene toda la información necesaria.

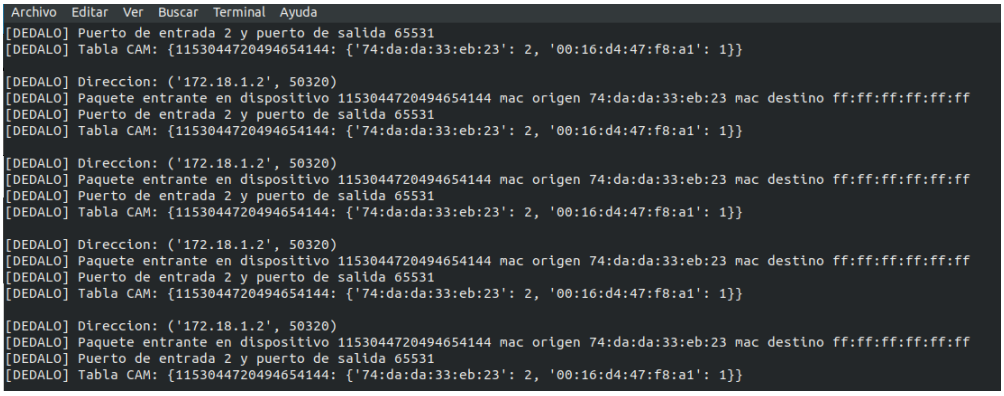

Figura 1.23: Mensajes del controlador del Prototipo 1

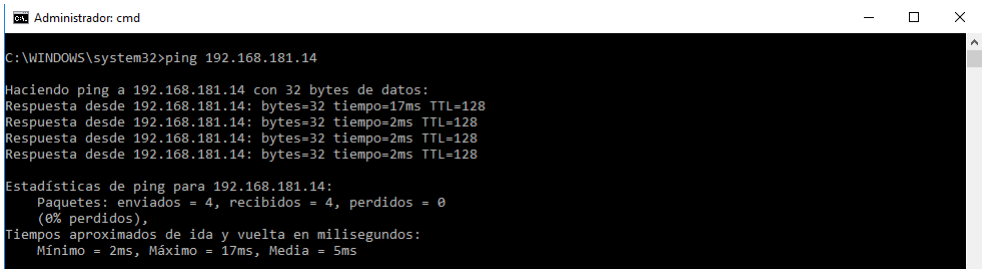

Figura 1.24: Conectividad entre Host 1 y Host 2 del Prototipo 1

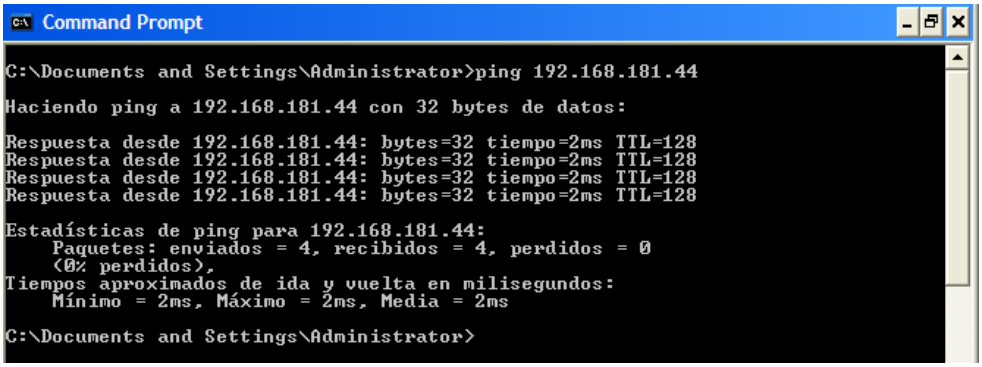

Figura 1.25: Conectividad entre Host 2 y Host 1 del Prototipo 1

1.10.4.2. Prototipo 2: Conexión de varios switches al mismo controlador

1.10.4.2.1 Objetivo

En esta iteración buscaremos implementar una conexión del controlador con tres conmutadores, de forma que todos ellos reciban la misma aplicación ryu y que cada uno gestione sus flujos correspondientes. Todo ello, con vistas a montar una topología un poco más compleja en el siguiente prototipo.

1.10.4.2.2 Implementación

En este caso, para que el controlador se vea más liberado y se produzcan menos latencias, se recomienda encarecidamente que cada conmutador guarde sus flujos. De esta forma, cada vez que entre un paquete, no lo mandará al controlador si tiene un flujo que dé información sobre qué hacer con él. En este caso, basta con añadir una función (add_flow) que lo haga.

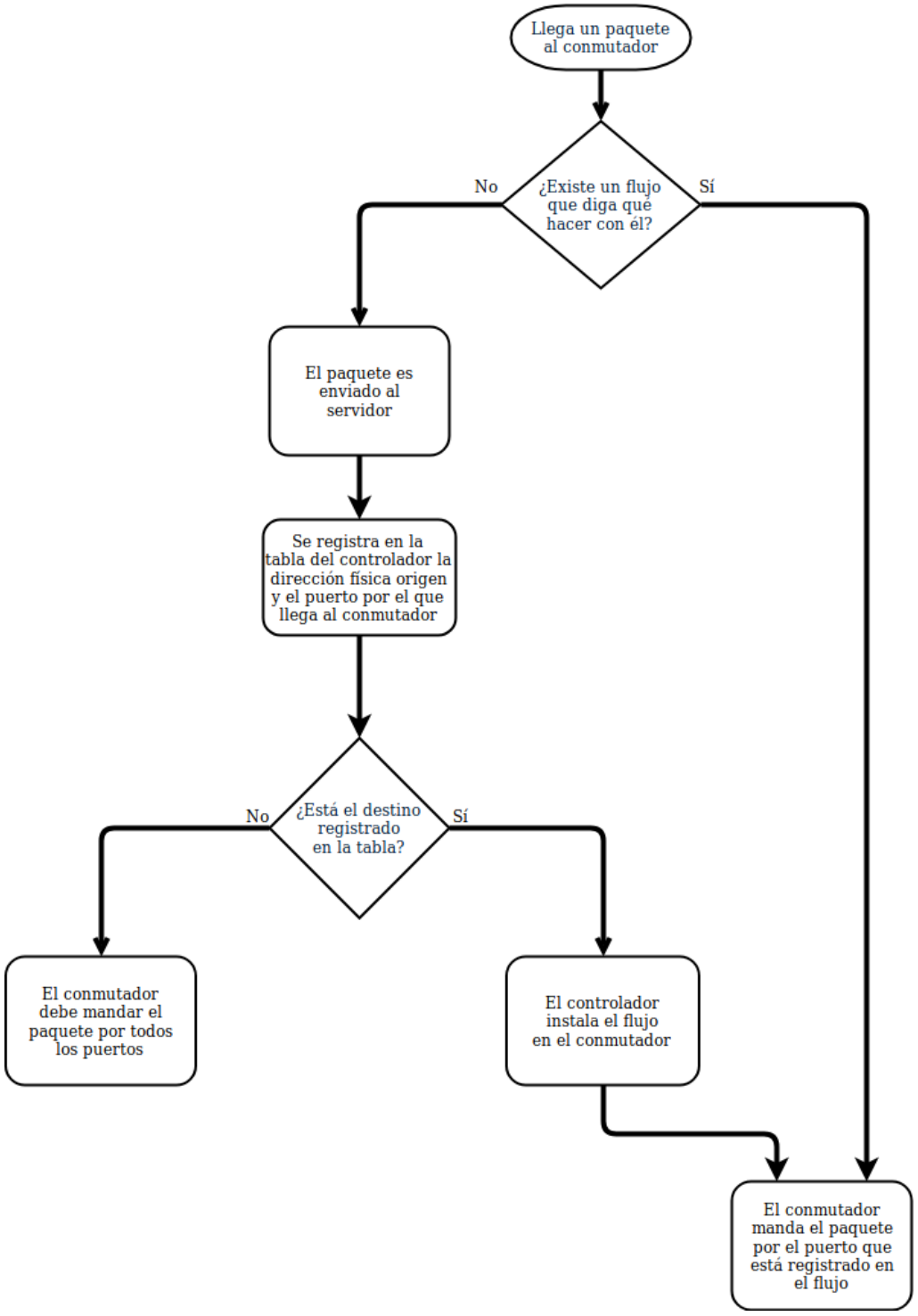

Figura 1.26: Diagrama de flujo del Prototipo 2

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_0
from ryu.lib.mac import haddr_to_bin
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
import sys

from ryu.lib.packet import packet, ethernet, arp, ipv4
import array

class SimpleSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]
    print("[DEDALO] Inicio aplicacion SimpleSwitch")

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    def add_flow(self, datapath, in_port, dst, src, actions):
        ofproto = datapath.ofproto

        match = datapath.ofproto_parser.OFPMatch(
            in_port=in_port,
            dl_dst=haddr_to_bin(dst), dl_src=haddr_to_bin(src))

        mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, match=match, cookie=0,
            command=ofproto.OFPFC_ADD, idle_timeout=0,
            hard_timeout=0,
            priority=ofproto.OFP_DEFAULT_PRIORITY,
            flags=ofproto.OFPFF_SEND_FLOW_REM, actions=actions)
        datapath.send_msg(mod)

    # Llega un paquete procedente del conmutador
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # Se desglosa el paquete en mensaje (msg), camino de datos
        # seguido (datapath) y protocolo OpenFlow tratado (ofproto)
        msg = ev.msg
```

```
        datapath = msg.datapath
        ofproto = datapath.ofproto

        self.logger.info("[DEDALO] Direccion: " +
            str(datapath.address))
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)

        # Obtencion de las direcciones fisicas origen y destino
        dst = eth.dst
        src = eth.src

        # La variable dpid contiene el identificador unico del
        # conmutador
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        # Se registra en la tabla la direccion fisica origen y
        # el puerto por el que llega
        self.mac_to_port[dpid][src] = msg.in_port

        # Esta el destino registrado en la tabla?
        # Si: El conmutador debe mandar el paquete por el puerto
        # out_port
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        # No: El conmutador debe mandar el paquete por todos los
        # puertos
        else:
            out_port = ofproto.OFPP_FLOOD

        actions =
            [datapath.ofproto_parser.OFPActionOutput(out_port)]

        # Instalamos el flujo si la salida no es una inundación
        if out_port != ofproto.OFPP_FLOOD:
            print("[DEDALO] Flujo instalado")
            self.add_flow(datapath, msg.in_port, dst, src, actions)

        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data
```

```
            print("[DEDALO] Paquete entrante en dispositivo " +
                str(dpid2) + " mac origen " + str(src2) +
                " mac destino " + str(dst2))
            print("[DEDALO] Puerto de entrada " +
                str(msg.in_port) + " y puerto de salida " +
                str(out_port))

            # Datos del paquete de salida
            out = datapath.ofproto_parser.OFPPacketOut(
                datapath=datapath,
                buffer_id=msg.buffer_id,
                in_port=msg.in_port,
                actions=actions, data=data)

            print("[DEDALO] Tabla CAM: " + str(self.mac_to_port) + "\n")

            # El paquete es enviado al conmutador
            datapath.send_msg(out)

    # Si se conecta o desconecta un equipo al conmutador, este lo
    # comunica al controlador para que lo saque o lo inserte
    # en su tabla
    @set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
    def _port_status_handler(self, ev):
        msg = ev.msg
        reason = msg.reason
        port_no = msg.desc.port_no

        ofproto = msg.datapath.ofproto
        if reason == ofproto.OFPPR_ADD:
            self.logger.info("puerto insertado %s", port_no)
        elif reason == ofproto.OFPPR_DELETE:
            self.logger.info("puerto eliminado %s", port_no)
        elif reason == ofproto.OFPPR_MODIFY:
            self.logger.info("puerto modificado %s", port_no)
        else:
            self.logger.info("Estado desconocido %s %s", port_no,
                             reason)
```

1.10.4.2.3 Simulación

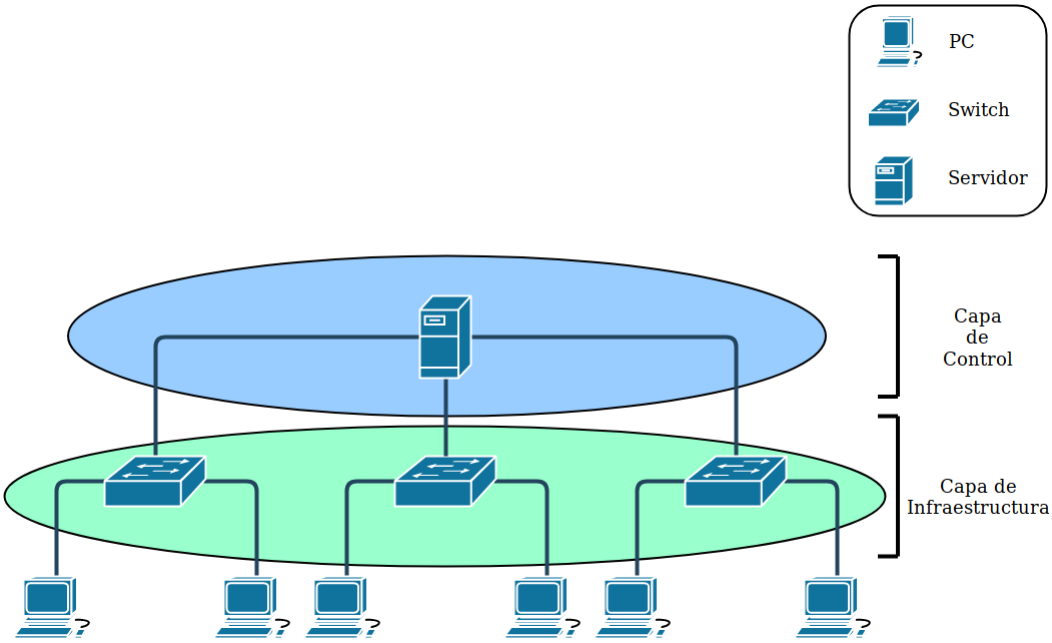

Figura 1.27: Topología simulada del Prototipo 2

Para implementar esta topología en mininet, se puede hacer uso del siguiente archivo python:

```
from mininet.cli import CLI
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.term import makeTerm

if '__main__' == __name__:
    net = Mininet(controller=RemoteController)

    c0 = net.addController('c0', port=6633)

    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')
    s3 = net.addSwitch('s3')

    h1 = net.addHost('h1')
    h2 = net.addHost('h2')
    h3 = net.addHost('h3')
```

```
h4 = net.addHost('h4')
h5 = net.addHost('h5')
h6 = net.addHost('h6')

net.addLink(s1, h1)
net.addLink(s1, h2)
net.addLink(s2, h3)
net.addLink(s2, h4)
net.addLink(s3, h5)
net.addLink(s3, h6)

net.build()
c0.start()
s1.start([c0])
s2.start([c0])
s3.start([c0])

CLI(net)

net.stop()
```

Tal y como hicimos en el prototipo anterior, lanzamos Mininet y Ryu a ejecución en dos consolas diferentes para obtener las siguientes pantallas. La primera mostrará que los `hosts` tienen conectividad entre ellos, no con los demás porque los conmutadores no están conectados entre ellos; y la segunda es la de Ryu y mostrará los mensajes de las funciones `print` que se han puesto en el código de la aplicación, añadiendo esta vez cuándo se está añadiendo un flujo.

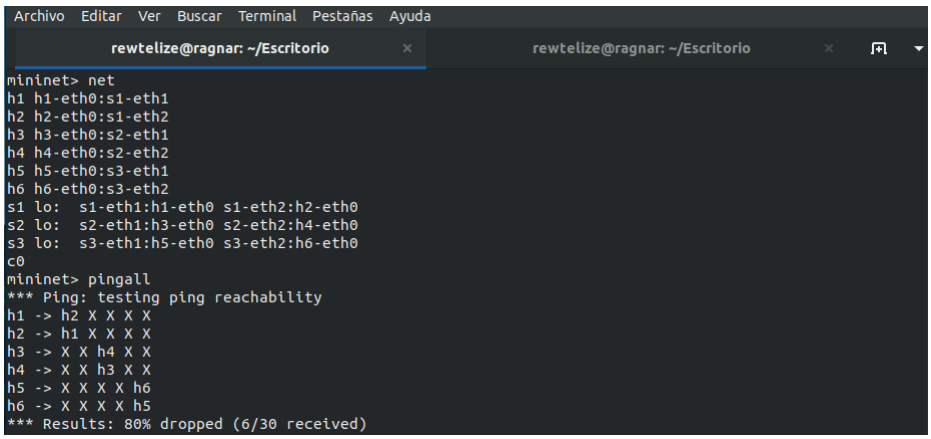

Figura 1.28: Simulación del Prototipo 2 - Parte I

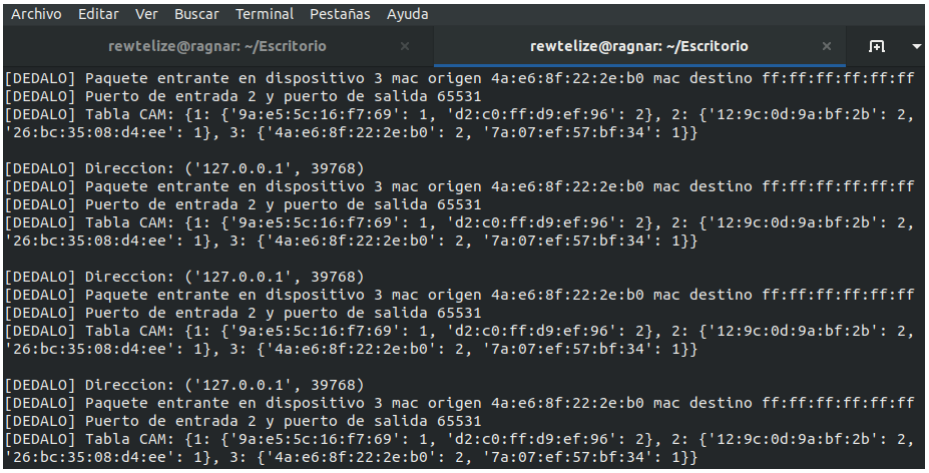

Figura 1.29: Simulación del Prototipo 2 - Parte II

1.10.4.2.4 Entorno real

A la topología empleada en el prototipo anterior, le añadimos dos conmutadores más y el blade:

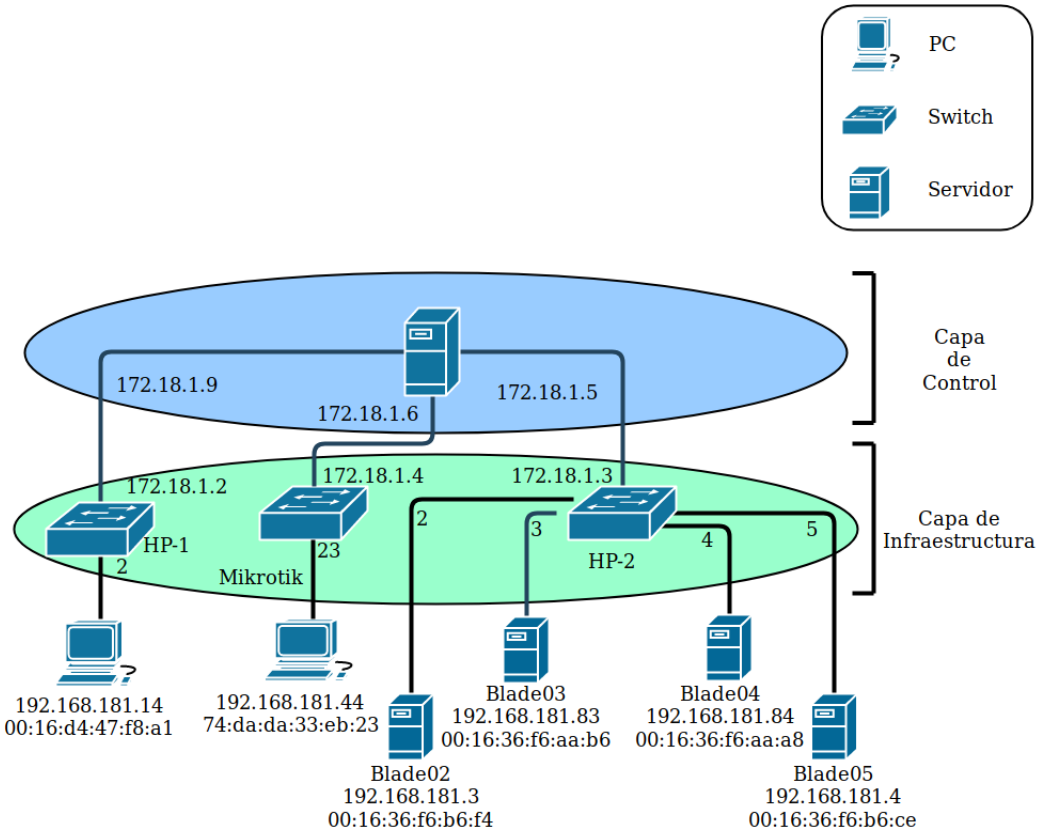

Figura 1.30: Topología real del Prototipo 2

En este prototipo tendremos que configurar los conmutadores HP-2 y MikroTik, ya que HP-1 fue configurado en el prototipo anterior.

```
S2(config)# openflow controller-id 1 ip 172.18.1.5 controller-interface vlan 50
S2(config)# openflow instance aggregate controller-id 1
S2(config)# openflow instance aggregate enable
S2(config)# openflow enable
```

```
[admin@MikroTik] >openflow add name=ofswitch1 controllers=172.18.1.6
[admin@MikroTik] >openflow enable 0
```

Nota: Para más información sobre los comandos empleados, véase Anexo 3.3. Configuración de los connmutadores.

Como en el prototipo anterior, lanzamos la aplicación ryu en la consola del servidor y comprobamos que la tabla del controlador contiene toda la información y que el conmutador HP-2, el cuál tiene más de un `host` conectado a él, tiene entradas en

su tabla de flujos.

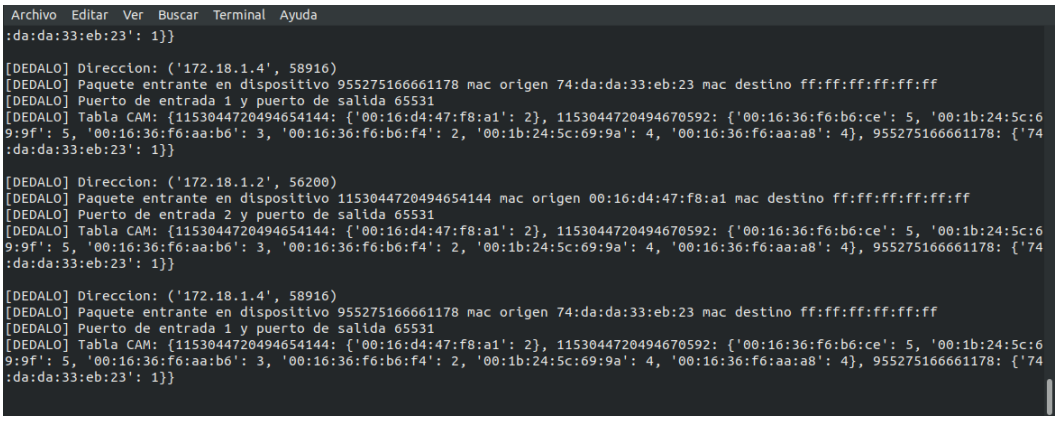

Figura 1.31: Mensajes del controlador del Prototipo 2

```
S2(config)# show openflow instance aggregate flows

 OpenFlow Flow Table

 Flow 1
 Match
  Incoming Port : 2                    Ethernet Type    : Any
  Source MAC    : 001636-f6b6f4        Destination MAC  : 001636-f6aab6
  VLAN ID       : Any                  VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                  IP ToS Bits      : Any
  Source Port   : Any                  Destination Port : Any
 Attributes
  Priority      : 32768                Duration         : 222 seconds
  Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
  Byte Count    : 332                  Packet Count     : 4
  Controller ID : 1                    Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                : 3
 Flow 2
 Match
  Incoming Port : 5                    Ethernet Type    : Any
  Source MAC    : 001636-f6b6ce        Destination MAC  : 001636-f6aab6
  VLAN ID       : Any                  VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                  IP ToS Bits      : Any
  Source Port   : Any                  Destination Port : Any
 Attributes
  Priority      : 32768                Duration         : 218 seconds
  Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
  Byte Count    : 332                  Packet Count     : 4
  Controller ID : 1                    Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                : 3
 Flow 3
 Match
  Incoming Port : 3                    Ethernet Type    : Any
  Source MAC    : 001636-f6aab6        Destination MAC  : 001636-f6b6ce
  VLAN ID       : Any                  VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                  IP ToS Bits      : Any
  Source Port   : Any                  Destination Port : Any
 Attributes
  Priority      : 32768                Duration         : 218 seconds
  Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
  Byte Count    : 332                  Packet Count     : 4
  Controller ID : 1                    Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                : 5

 Flow 4
 Match
  Incoming Port : 4                    Ethernet Type    : Any
  Source MAC    : 001636-f6aaa8        Destination MAC  : 001636-f6aab6
  VLAN ID       : Any                  VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                  IP ToS Bits      : Any
  Source Port   : Any                  Destination Port : Any
 Attributes
  Priority      : 32768                Duration         : 228 seconds
  Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
  Byte Count    : 434                  Packet Count     : 5
  Controller ID : 1                    Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                : 3
```

Figura 1.32: Flujos de HP-2 del Prototipo 2 - Parte I

```
Flow 5
Match
 Incoming Port : 3                    Ethernet Type    : Any
 Source MAC    : 001636-f6aab6        Destination MAC  : 001636-f6b6f4
 VLAN ID       : Any                  VLAN priority    : Any
 Source Protocol Address : Any
 Target Protocol Address : Any
 IP Protocol   : Any                  IP ToS Bits      : Any
 Source Port   : Any                  Destination Port : Any
Attributes
 Priority      : 32768                Duration         : 221 seconds
 Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
 Byte Count    : 230                  Packet Count     : 3
 Controller ID : 1                    Cookie           : 0x0
 Flow Location : Software
 Hardware Index     : NA
 Reason Code        : 15
 Reason Description : Rule cannot be accelerated in hardware
Actions
   Output                : 2

Flow 6
Match
 Incoming Port : 3                    Ethernet Type    : Any
 Source MAC    : 001636-f6aab6        Destination MAC  : 001636-f6aaa8
 VLAN ID       : Any                  VLAN priority    : Any
 Source Protocol Address : Any
 Target Protocol Address : Any
 IP Protocol   : Any                  IP ToS Bits      : Any
 Source Port   : Any                  Destination Port : Any
Attributes
 Priority      : 32768                Duration         : 227 seconds
 Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
 Byte Count    : 332                  Packet Count     : 4
 Controller ID : 1                    Cookie           : 0x0
 Flow Location : Software
 Hardware Index     : NA
 Reason Code        : 15
 Reason Description : Rule cannot be accelerated in hardware
Actions
   Output                : 4
```

Figura 1.33: Flujos de HP-2 del Prototipo 2 - Parte II

1.10.4.3. Prototipo 3: Interconexión de switches en forma de anillo

1.10.4.3.1 Objetivo

El objetivo en este prototipo es montar un topología en anillo. Para ello, es necesario una implementación cercana al protocolo Spanning Tree Protocol (STP). De esta manera, estaríamos simulando el área metropolitana de la ciudad de Cádiz con una arquitectura SDN.

1.10.4.3.2 Implementación

Para añadir el STP a nuestro código, se ha optado por definir los distintos estados que puede tener un puerto:

- Deshabilitado (DISABLE): El puerto está deshabilitado por el administrador de red.
- Bloqueado (BLOCK): El puerto ni envía ni recibe tramas y queda en ese estado hasta que el conmutador decida que hay un camino mejor y cuente con él.
- Escuchando (LISTEN): Estado transitorio por el que pasa el puerto para asegurar que forma parte de una ruta sin bucles.

- Aprendiendo (LEARN): Segundo estado transitorio, en el cual aprende direcciones físicas pero no reenvía tramas.
- Enviando (FORWARD): El puerto envía y recibe tramas.

En la implementación, estos estados sólo pueden cambiar si se recibe un evento de cambio de estado de puertos (EventPortStateChange)

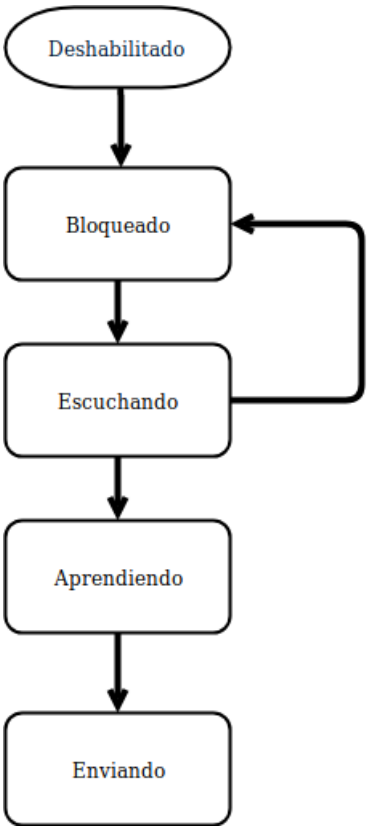

Figura 1.34: Diagrama de flujo del Prototipo 3: Puertos

Por otro lado, como queremos una sistema dinámico y automatizado, hemos de tener en cuenta que cuando se añada un nuevo conmutador a la topología, esta puede variar y necesitará desechar flujos. Por esta razón, se incluye una función de eliminación de flujos (delete_flow), y cuando se detecte un evento de cambio de topología (EventTopologyChange) se reiniciará las tablas CAM.

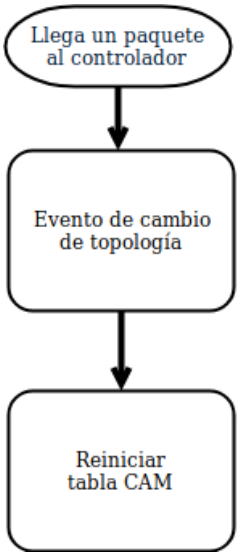

Figura 1.35: Diagrama de flujo del Prototipo 3: Flujos

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_0
from ryu.lib.mac import haddr_to_bin
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
import sys

from ryu.lib.packet import packet, ethernet, arp, ipv4
import array

from ryu.lib import stplib
from ryu.lib import dpid as dpid_lib

class SimpleSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]
    _CONTEXTS = {'stplib': stplib.Stp}
    print("[DEDALO] Inicio aplicacion SimpleSwitch")

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
```

```
        self.stp = kwargs['stplib']

    def add_flow(self, datapath, in_port, dst, src, actions):
        print("[DEDALO] Nuevo flujo")
        ofproto = datapath.ofproto

        match = datapath.ofproto_parser.OFPMatch(
            in_port=in_port,
            dl_dst=haddr_to_bin(dst), dl_src=haddr_to_bin(src))

        mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, match=match, cookie=0,
            command=ofproto.OFPFC_ADD, idle_timeout=0,
            hard_timeout=0,
            priority=ofproto.OFP_DEFAULT_PRIORITY,
            flags=ofproto.OFPFF_SEND_FLOW_REM, actions=actions)
        datapath.send_msg(mod)

    def delete_flow(self, datapath):
        print("[DEDALO] Eliminacion de flujo")
        ofproto = datapath.ofproto

        wildcards = ofproto_v1_0.OFPFW_ALL
        match = datapath.ofproto_parser.OFPMatch(
            wildcards, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

        mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, match=match, cookie=0,
            command=ofproto.OFPFC_DELETE)
        datapath.send_msg(mod)

    # Llega un paquete procedente del conmutador
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # Se desglosa el paquete en mensaje (msg), camino de datos
        # seguido (datapath) y protocolo OpenFlow tratado (ofproto)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto

        self.logger.info("[DEDALO] Direccion: " +
            str(datapath.address))
        pkt = packet.Packet(msg.data)
```

```
        eth = pkt.get_protocol(ethernet.ethernet)

        # Obtencion de las direcciones fisicas origen y destino
        dst = eth.dst
        src = eth.src

        # La variable dpid contiene el identificador unico del
        # conmutador
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        # Se registra en la tabla la direccion fisica origen y
        # el puerto por el que llega
        self.mac_to_port[dpid][src] = msg.in_port

        # Esta el destino registrado en la tabla?
        # Si: El conmutador debe mandar el paquete por el puerto
        # out_port
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        # No: El conmutador debe mandar el paquete por todos los
        # puertos
        else:
            out_port = ofproto.OFPP_FLOOD

        actions =
            [datapath.ofproto_parser.OFPActionOutput(out_port)]

        # Instalamos el flujo si la salida no es una inundación
        if out_port != ofproto.OFPP_FLOOD:
            print("[DEDALO] Flujo instalado")
            self.add_flow(datapath, msg.in_port, dst, src, actions)

        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        print("[DEDALO] Paquete entrante en dispositivo " +
            str(dpid2) + " mac origen " + str(src2) +
            " mac destino " + str(dst2))
        print("[DEDALO] Puerto de entrada " +
            str(msg.in_port) + " y puerto de salida " +
            str(out_port))
```

```
        # Datos del paquete de salida
        out = datapath.ofproto_parser.OFPPacketOut(
            datapath=datapath,
            buffer_id=msg.buffer_id,
            in_port=msg.in_port,
            actions=actions, data=data)

        print("[DEDALO] Tabla CAM: " + str(self.mac_to_port) + "\n")

        # El paquete es enviado al conmutador
        datapath.send_msg(out)

    # Si se conecta o desconecta un equipo al conmutador, este lo
    # comunica al controlador para que lo saque o lo inserte
    # en su tabla
    @set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
    def _port_status_handler(self, ev):
        msg = ev.msg
        reason = msg.reason
        port_no = msg.desc.port_no

        ofproto = msg.datapath.ofproto
        if reason == ofproto.OFPPR_ADD:
            self.logger.info("puerto insertado %s", port_no)
        elif reason == ofproto.OFPPR_DELETE:
            self.logger.info("puerto eliminado %s", port_no)
        elif reason == ofproto.OFPPR_MODIFY:
            self.logger.info("puerto modificado %s", port_no)
        else:
            self.logger.info("Estado desconocido %s %s", port_no,
                             reason)

    @set_ev_cls(stplib.EventTopologyChange, MAIN_DISPATCHER)
    def _topology_change_handler(self, ev):
        dp = ev.dp
        dpid_str = dpid_lib.dpid_to_str(dp.id)
        print("[DEDALO][Topologia][dpid=" + str(dpid_str) +
            "] Vaciado de TCAM")

        if dp.id in self.mac_to_port:
            del self.mac_to_port[dp.id]
        self.delete_flow(dp)
```

```
@set_ev_cls(stplib.EventPortStateChange, MAIN_DISPATCHER)
def _port_state_change_handler(self, ev):
    dpid_str = dpid_lib.dpid_to_str(ev.dp.id)
    of_state = {stplib.PORT_STATE_DISABLE: 'DISABLE',
                stplib.PORT_STATE_BLOCK: 'BLOCK',
                stplib.PORT_STATE_LISTEN: 'LISTEN',
                stplib.PORT_STATE_LEARN: 'LEARN',
                stplib.PORT_STATE_FORWARD: 'FORWARD'}
    self.logger.debug("[dpid=%s][port=%d] state=%s",
        dpid_str, ev.port_no, of_state[ev.port_state])
```

1.10.4.3.3 Simulación

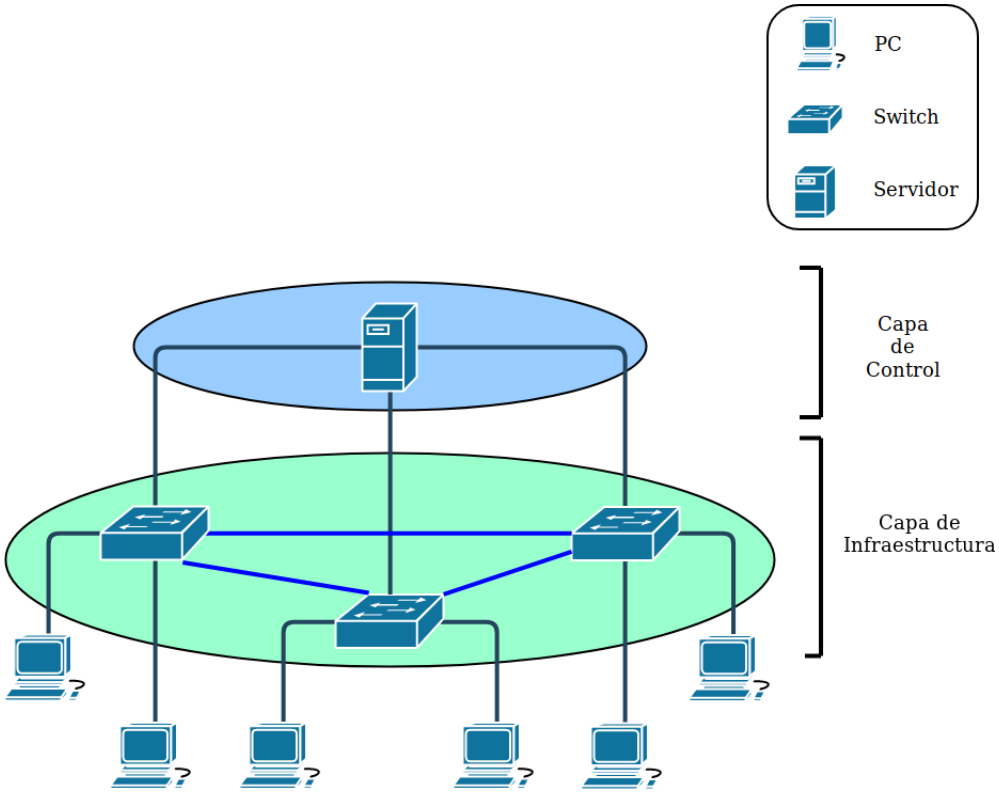

Figura 1.36: Topología simulada del Prototipo 3

Para simular esta topología, añadiremos las conexiones entre los conmutadores del archivo python de la topología anterior:

```
from mininet.cli import CLI
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.term import makeTerm

if '__main__' == __name__:
    net = Mininet(controller=RemoteController)

    c0 = net.addController('c0', port=6633)

    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')
    s3 = net.addSwitch('s3')

    h1 = net.addHost('h1')
    h2 = net.addHost('h2')
    h3 = net.addHost('h3')
    h4 = net.addHost('h4')
    h5 = net.addHost('h5')
    h6 = net.addHost('h6')

    net.addLink(s1, h1)
    net.addLink(s1, h2)
    net.addLink(s2, h3)
    net.addLink(s2, h4)
    net.addLink(s3, h5)
    net.addLink(s3, h6)

    net.addLink(s1, s2)
    net.addLink(s2, s3)
    net.addLink(s3, s1)

    net.build()
    c0.start()
    s1.start([c0])
    s2.start([c0])
    s3.start([c0])

    CLI(net)

    net.stop()
```

Y, como siempre, lanzamos Mininet y Ryu a ejecución en dos consolas diferentes para obtener las siguientes pantallas. La primera mostrará que los `hosts` tienen

conectividad y que no se están perdiendo paquetes en ningún bucle; y la segunda es la de Ryu y mostrará los mensajes de las funciones `print` que se han puesto en prototipos anteriores y mensajes del protocolo STP que se han añadido nuevos.

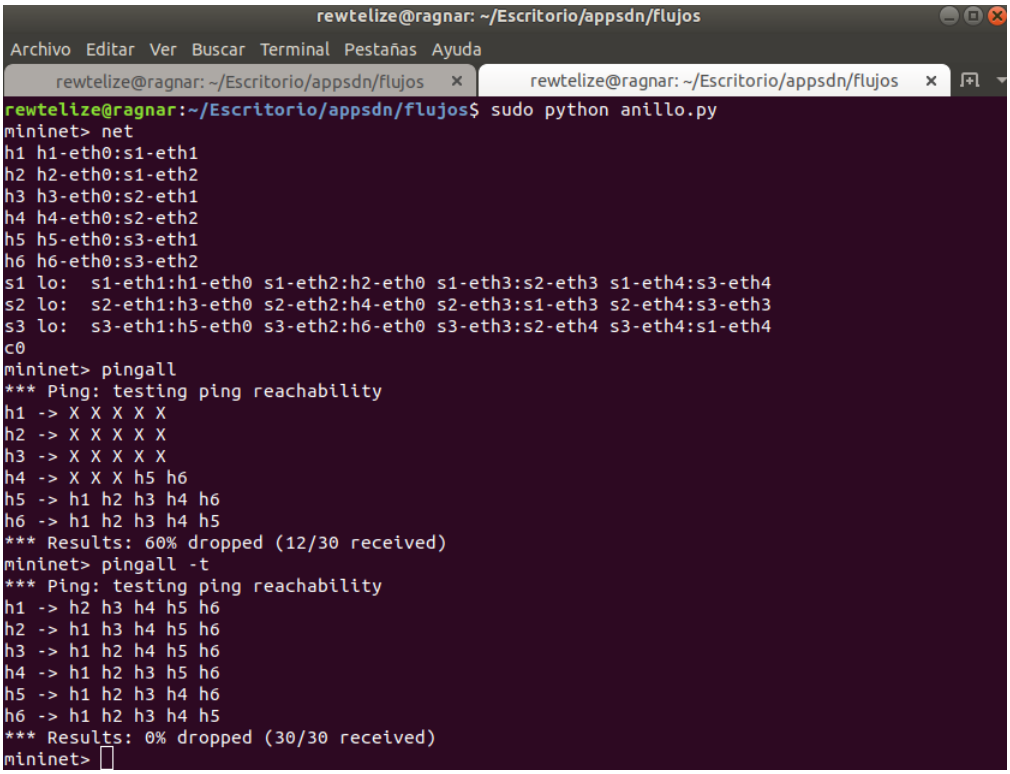

Figura 1.37: Simulación del Prototipo 3 - Parte I

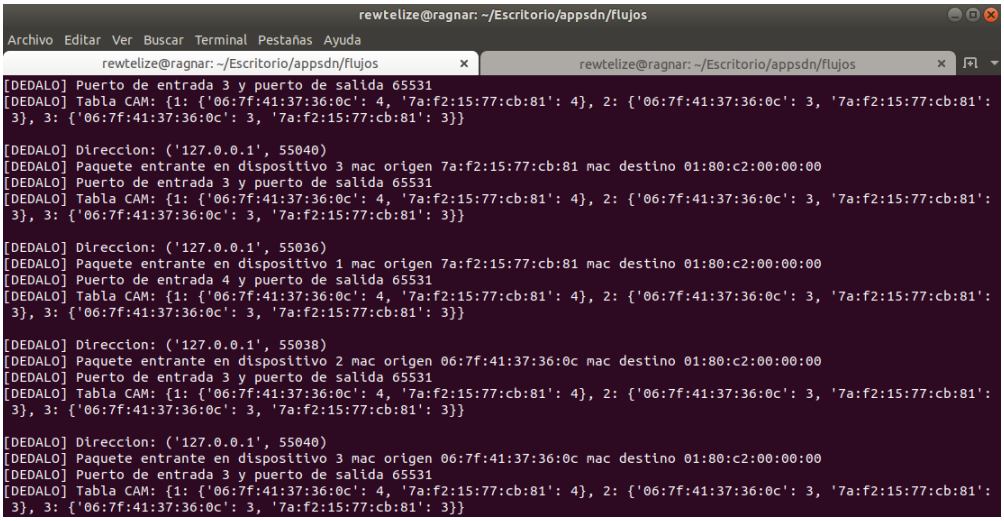

Figura 1.38: Simulación del Prototipo 3 - Parte II

1.10.4.3.4 Entorno real

Con respecto al prototipo anterior, para montar la topología únicamente es necesario interconectar los tres conmutadores.

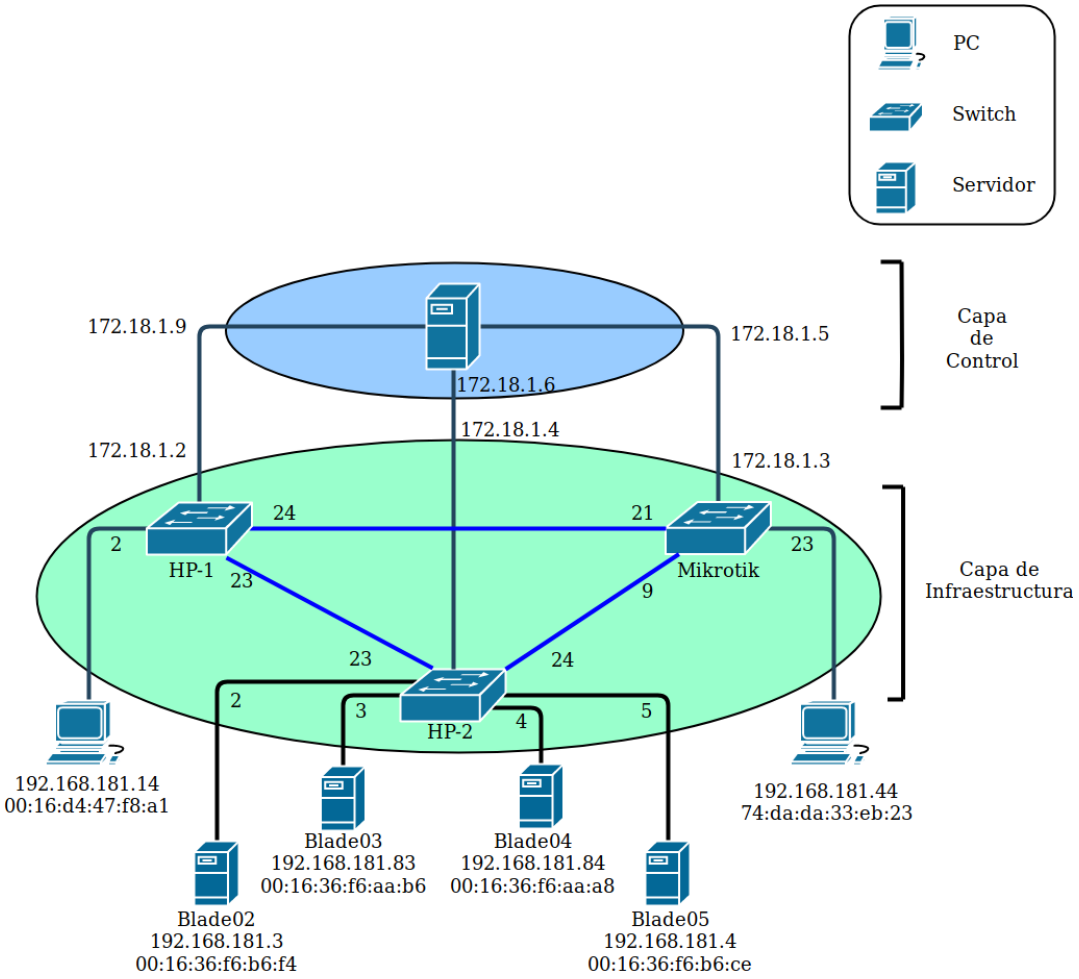

Figura 1.39: Topología real del Prototipo 3

Puesto que los conmutadores ya están configurados para trabajar con OpenFlow, pasamos directamente a lanzar la aplicación ryu en la consola del servidor y comprobamos que la tabla del controlador contiene toda la información, que cada conmutador almacena los flujos necesarios y que los `hosts` tienen conectividad.

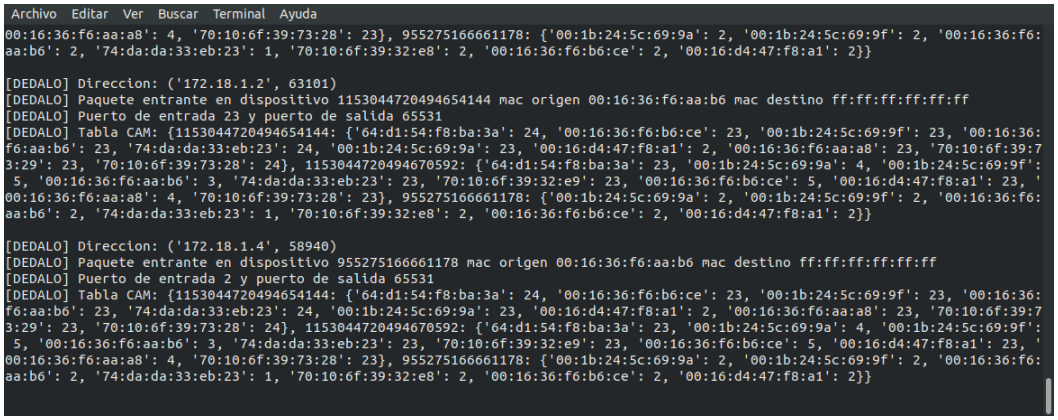

Figura 1.40: Mensajes del controlador del Prototipo 3

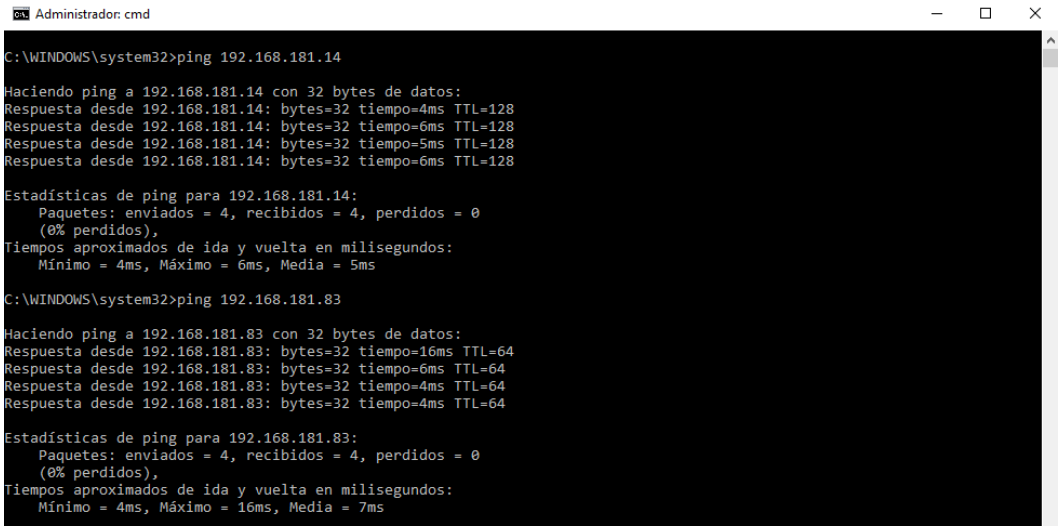

Figura 1.41: Conectividad entre hosts de Mikrotik con HP-1 y HP-2 del Prototipo 3

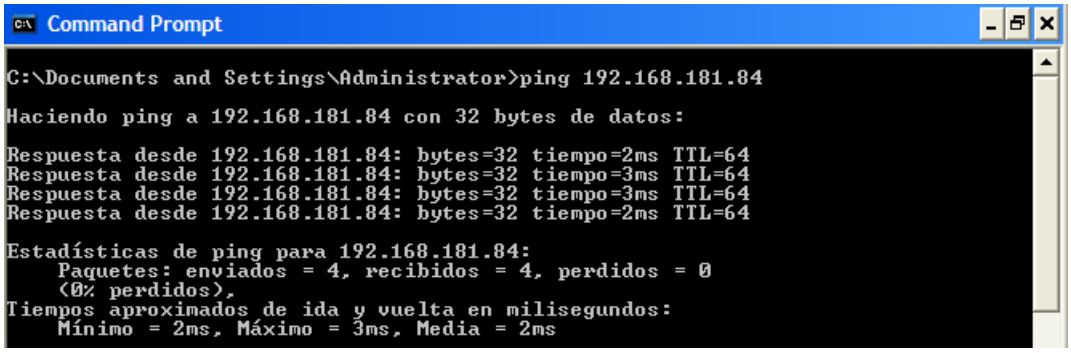

Figura 1.42: Conectividad entre hosts de HP-1 con HP-2 del Prototipo 3

```
S1(config)# show openflow instance aggregate flows

 OpenFlow Flow Table

 Flow 1
 Match
  Incoming Port : 24                    Ethernet Type    : Any
  Source MAC    : 74dada-33eb23         Destination MAC  : 001636-f6aab6
  VLAN ID       : Any                   VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                   IP ToS Bits      : Any
  Source Port   : Any                   Destination Port : Any
 Attributes
  Priority      : 32768                 Duration         : 362 seconds
  Hard Timeout  : 0 seconds             Idle Timeout     : 0 seconds
  Byte Count    : 1000                  Packet Count     : 13
  Controller ID : 1                     Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                : 23
 Flow 2
 Match
  Incoming Port : 23                    Ethernet Type    : Any
  Source MAC    : 001636-f6aaa8         Destination MAC  : 0016d4-47f8a1
  VLAN ID       : Any                   VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                   IP ToS Bits      : Any
  Source Port   : Any                   Destination Port : Any
 Attributes
  Priority      : 32768                 Duration         : 312 seconds
  Hard Timeout  : 0 seconds             Idle Timeout     : 0 seconds
  Byte Count    : 610                   Packet Count     : 8
  Controller ID : 1                     Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                : 2
 Flow 3
 Match
  Incoming Port : 24                    Ethernet Type    : Any
  Source MAC    : 74dada-33eb23         Destination MAC  : 0016d4-47f8a1
  VLAN ID       : Any                   VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                   IP ToS Bits      : Any
  Source Port   : Any                   Destination Port : Any
 Attributes
  Priority      : 32768                 Duration         : 345 seconds
  Hard Timeout  : 0 seconds             Idle Timeout     : 0 seconds
  Byte Count    : 532                   Packet Count     : 7
  Controller ID : 1                     Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                : 2

 Flow 4
 Match
  Incoming Port : 2                     Ethernet Type    : Any
  Source MAC    : 0016d4-47f8a1         Destination MAC  : 74dada-33eb23
  VLAN ID       : Any                   VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                   IP ToS Bits      : Any
  Source Port   : Any                   Destination Port : Any
 Attributes
  Priority      : 32768                 Duration         : 346 seconds
  Hard Timeout  : 0 seconds             Idle Timeout     : 0 seconds
  Byte Count    : 610                   Packet Count     : 8
  Controller ID : 1                     Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
```

Figura 1.43: Flujos de HP-1 del Prototipo 3 - Parte I

```
Flow 5
Match
 Incoming Port : 2                    Ethernet Type    : Any
 Source MAC    : 0016d4-47f8a1        Destination MAC  : 001636-f6aaa8
 VLAN ID       : Any                  VLAN priority    : Any
 Source Protocol Address : Any
 Target Protocol Address : Any
 IP Protocol   : Any                  IP ToS Bits      : Any
 Source Port   : Any                  Destination Port : Any
Attributes
 Priority      : 32768                Duration         : 311 seconds
 Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
 Byte Count    : 532                  Packet Count     : 7
 Controller ID : 1                    Cookie           : 0x0
 Flow Location : Software
 Hardware Index     : NA
 Reason Code        : 15
 Reason Description : Rule cannot be accelerated in hardware
Actions
   Output                : 23

Flow 6
Match
 Incoming Port : 23                   Ethernet Type    : Any
 Source MAC    : 001636-f6aab6        Destination MAC  : 74dada-33eb23
 VLAN ID       : Any                  VLAN priority    : Any
 Source Protocol Address : Any
 Target Protocol Address : Any
 IP Protocol   : Any                  IP ToS Bits      : Any
 Source Port   : Any                  Destination Port : Any
Attributes
 Priority      : 32768                Duration         : 363 seconds
 Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
 Byte Count    : 1078                 Packet Count     : 14
 Controller ID : 1                    Cookie           : 0x0
 Flow Location : Software
 Hardware Index     : NA
 Reason Code        : 15
 Reason Description : Rule cannot be accelerated in hardware
Actions
   Output                : 24
```

Figura 1.44: Flujos de HP-1 del Prototipo 3 - Parte II

```
S2(config)# show openflow instance aggregate flows

 OpenFlow Flow Table

 Flow 1
 Match
  Incoming Port : 4                    Ethernet Type    : Any
  Source MAC    : 001636-f6aaa8        Destination MAC  : 0016d4-47f8a1
  VLAN ID       : Any                  VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                  IP ToS Bits      : Any
  Source Port   : Any                  Destination Port : Any
 Attributes
  Priority      : 32768                Duration         : 308 seconds
  Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
  Byte Count    : 610                  Packet Count     : 8
  Controller ID : 1                    Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                : 23
 Flow 2
 Match
  Incoming Port : 23                   Ethernet Type    : Any
  Source MAC    : 74dada-33eb23        Destination MAC  : 001636-f6aab6
  VLAN ID       : Any                  VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                  IP ToS Bits      : Any
  Source Port   : Any                  Destination Port : Any
 Attributes
  Priority      : 32768                Duration         : 330 seconds
  Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
  Byte Count    : 234                  Packet Count     : 3
  Controller ID : 1                    Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                : 3
Flow 3
Match
 Incoming Port : 3                    Ethernet Type    : Any
 Source MAC    : 001636-f6aab6        Destination MAC  : 74dada-33eb23
 VLAN ID       : Any                  VLAN priority    : Any
 Source Protocol Address : Any
 Target Protocol Address : Any
 IP Protocol   : Any                  IP ToS Bits      : Any
 Source Port   : Any                  Destination Port : Any
Attributes
 Priority      : 32768                Duration         : 330 seconds
 Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
 Byte Count    : 234                  Packet Count     : 3
 Controller ID : 1                    Cookie           : 0x0
 Flow Location : Software
 Hardware Index     : NA
 Reason Code        : 15
 Reason Description : Rule cannot be accelerated in hardware
Actions
   Output                : 23

Flow 4
Match
 Incoming Port : 23                   Ethernet Type    : Any
 Source MAC    : 0016d4-47f8a1        Destination MAC  : 001636-f6aaa8
 VLAN ID       : Any                  VLAN priority    : Any
 Source Protocol Address : Any
 Target Protocol Address : Any
 IP Protocol   : Any                  IP ToS Bits      : Any
 Source Port   : Any                  Destination Port : Any
Attributes
 Priority      : 32768                Duration         : 307 seconds
 Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
 Byte Count    : 532                  Packet Count     : 7
 Controller ID : 1                    Cookie           : 0x0
 Flow Location : Software
 Hardware Index     : NA
 Reason Code        : 15
 Reason Description : Rule cannot be accelerated in hardware
Actions
   Output                : 4
```

Figura 1.45: Flujos de HP-2 del Prototipo 3

```
[admin@MikroTik] > openflow flow print
Flags: I - inactive
 #    SWITCH                 MATCH                    ACTIONS
 0    ofswitch1              inport:2 dlsrc:00:16...  output:1
 1    ofswitch1              inport:1 dlsrc:74:DA...  output:2
[admin@MikroTik] >
```

Figura 1.46: Flujos de Mikrotik del Prototipo 3

1.10.4.4. Prototipo 4: Implementación de políticas

1.10.4.4.1 Objetivo

Con el anillo ya implementado, buscamos a continuación implementar uno de los requisitos recogidos en la especificación: las políticas. Un usuario debe ser capaz de decirle al sistema que paquetes debe rechazar, a partir de un puerto origen, un puerto destino y/o un protocolo.

1.10.4.4.2 Implementación

En este caso, no ha sido necesario hacer grandes modificaciones en el código, ya que basta con poner una condición antes de enviar los paquetes. Como se ha mencionado en apartados anteriores, pondremos la condición de que si el puerto origen, el puerto destino o el protocolo coincide con los que el administrador ha incluído en la politica, entonces el paquete no saldrá del servidor y tampoco se instalará como flujo.

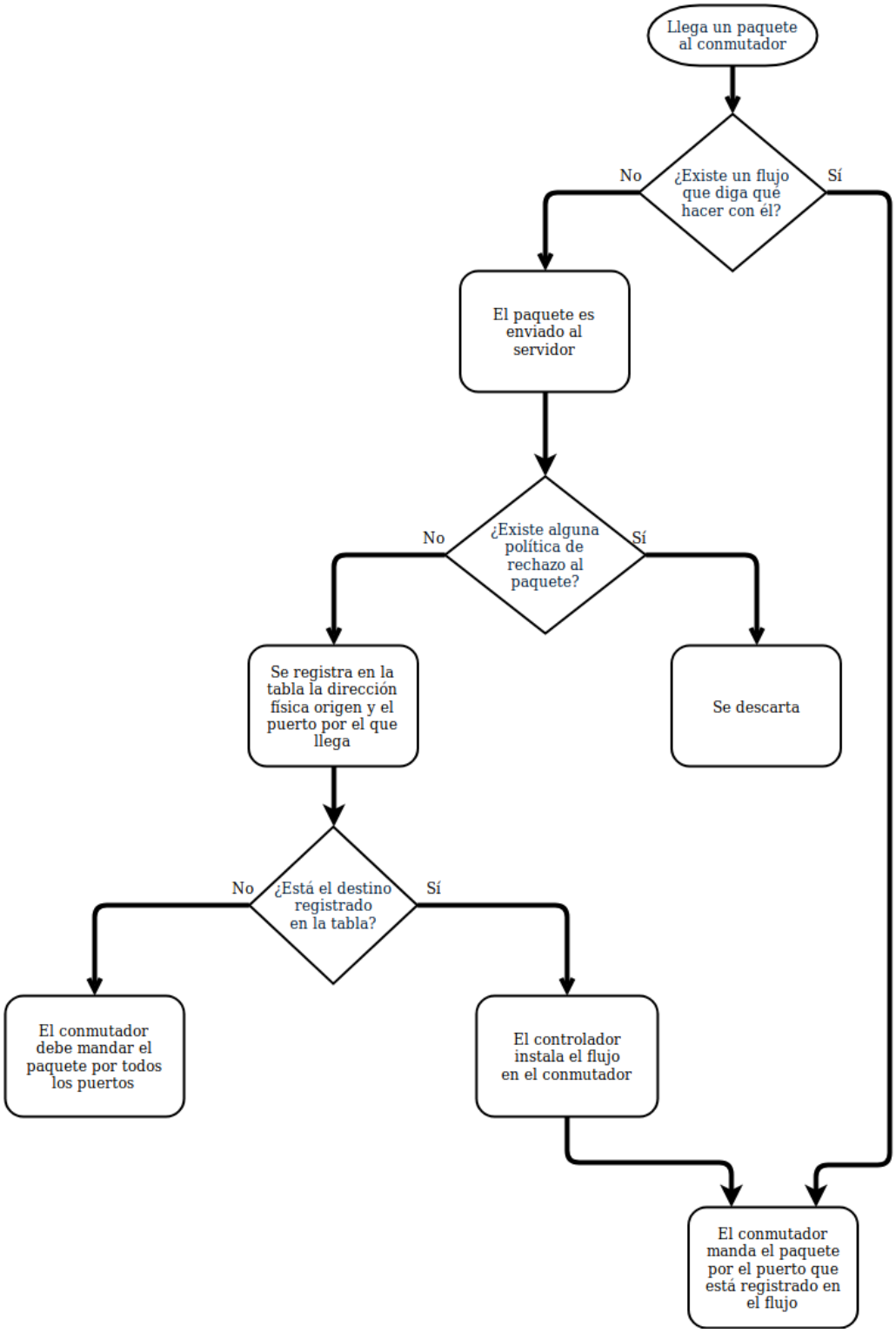

Figura 1.47: Diagrama de flujo del Prototipo 4

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_0
from ryu.lib.mac import haddr_to_bin
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
import sys

from ryu.lib.packet import packet, ethernet, arp, ipv4
import array

from ryu.lib import stplib
from ryu.lib import dpid as dpid_lib

class SimpleSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]
    _CONTEXTS = {'stplib': stplib.Stp}
    print("[DEDALO] Inicio aplicacion SimpleSwitch")

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.stp = kwargs['stplib']

    def add_flow(self, datapath, in_port, dst, src, actions):
        print("[DEDALO] Nuevo flujo")
        ofproto = datapath.ofproto

        match = datapath.ofproto_parser.OFPMatch(
            in_port=in_port,
            dl_dst=haddr_to_bin(dst), dl_src=haddr_to_bin(src))

        mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, match=match, cookie=0,
            command=ofproto.OFPFC_ADD, idle_timeout=0,
            hard_timeout=0,
            priority=ofproto.OFP_DEFAULT_PRIORITY,
            flags=ofproto.OFPFF_SEND_FLOW_REM, actions=actions)
        datapath.send_msg(mod)
```

```
    def delete_flow(self, datapath):
        print("[DEDALO] Eliminacion de flujo")
        ofproto = datapath.ofproto

        wildcards = ofproto_v1_0.OFPFW_ALL
        match = datapath.ofproto_parser.OFPMatch(
            wildcards, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

        mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, match=match, cookie=0,
            command=ofproto.OFPFC_DELETE)
        datapath.send_msg(mod)

    # Llega un paquete procedente del conmutador
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # Se desglosa el paquete en mensaje (msg), camino de datos
        # seguido (datapath) y protocolo OpenFlow tratado (ofproto)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto

        self.logger.info("[DEDALO] Direccion: " +
            str(datapath.address))
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)

        # Obtencion de las direcciones fisicas origen y destino
        dst = eth.dst
        src = eth.src

        # La variable dpid contiene el identificador unico del
        # conmutador
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        # Se registra en la tabla la direccion fisica origen y
        # el puerto por el que llega
        if msg.in_port != 2 and msg.in_port != 21:
            self.mac_to_port[dpid][src] = msg.in_port

            # Esta el destino registrado en la tabla?
            # Si: El conmutador debe mandar el paquete por el puerto
```

```
            # out_port
            if dst in self.mac_to_port[dpid]:
                out_port = self.mac_to_port[dpid][dst]
            # No: El conmutador debe mandar el paquete por todos los
            # puertos
            else:
                out_port = ofproto.OFPP_FLOOD

            actions =
                [datapath.ofproto_parser.OFPActionOutput(out_port)]

            # Instalamos el flujo si la salida no es una inundación
            if out_port != ofproto.OFPP_FLOOD:
                print("[DEDALO] Flujo instalado")
                self.add_flow(datapath, msg.in_port, dst, src,
                              actions)

            data = None
            if msg.buffer_id == ofproto.OFP_NO_BUFFER:
                data = msg.data

            print("[DEDALO] Paquete entrante en dispositivo " +
                str(dpid2) + " mac origen " + str(src2) +
                " mac destino " + str(dst2))
            print("[DEDALO] Puerto de entrada " +
                str(msg.in_port) + " y puerto de salida " +
                str(out_port))

            # Datos del paquete de salida
            out = datapath.ofproto_parser.OFPPacketOut(
                datapath=datapath,
                buffer_id=msg.buffer_id,
                in_port=msg.in_port,
                actions=actions, data=data)

            print("[DEDALO] Tabla CAM: " + str(self.mac_to_port)
                    + "\n")

            # El paquete es enviado al conmutador
            datapath.send_msg(out)

    # Si se conecta o desconecta un equipo al conmutador, este lo
    # comunica al controlador para que lo saque o lo inserte
```

```
    # en su tabla
    @set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
    def _port_status_handler(self, ev):
        msg = ev.msg
        reason = msg.reason
        port_no = msg.desc.port_no

        ofproto = msg.datapath.ofproto
        if reason == ofproto.OFPPR_ADD:
            self.logger.info("puerto insertado %s", port_no)
        elif reason == ofproto.OFPPR_DELETE:
            self.logger.info("puerto eliminado %s", port_no)
        elif reason == ofproto.OFPPR_MODIFY:
            self.logger.info("puerto modificado %s", port_no)
        else:
            self.logger.info("Estado desconocido %s %s", port_no,
                             reason)

    @set_ev_cls(stplib.EventTopologyChange, MAIN_DISPATCHER)
    def _topology_change_handler(self, ev):
        dp = ev.dp
        dpid_str = dpid_lib.dpid_to_str(dp.id)
        print("[DEDALO][Topologia][dpid=" + str(dpid_str) +
            "] Vaciado de TCAM")

        if dp.id in self.mac_to_port:
            del self.mac_to_port[dp.id]
        self.delete_flow(dp)

    @set_ev_cls(stplib.EventPortStateChange, MAIN_DISPATCHER)
    def _port_state_change_handler(self, ev):
        dpid_str = dpid_lib.dpid_to_str(ev.dp.id)
        of_state = {stplib.PORT_STATE_DISABLE: 'DISABLE',
                    stplib.PORT_STATE_BLOCK: 'BLOCK',
                    stplib.PORT_STATE_LISTEN: 'LISTEN',
                    stplib.PORT_STATE_LEARN: 'LEARN',
                    stplib.PORT_STATE_FORWARD: 'FORWARD'}
        self.logger.debug("[dpid=%s][port=%d] state=%s",
            dpid_str, ev.port_no, of_state[ev.port_state])
```

1.10.4.4.3 Simulación

Se muestra una posible implementación en el que el controlador responderá a aquellos conmutadores que pregunten por paquetes cuyos puertos de entrada sean distintos del dos y del ventiuno. Como se muestra a continuación, los puertos número dos corresponden con los `hosts` pares de cada conmutador. Luego sólo tendrán conectividad los dispositivos h1, h3 y h5.

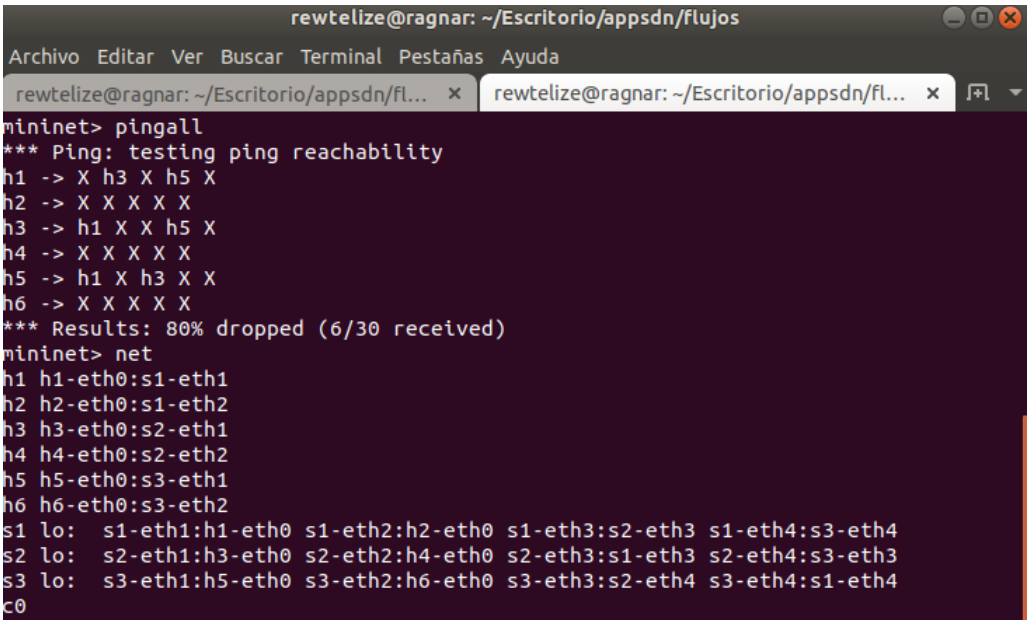

Figura 1.48: Simulación del Prototipo 4 - Parte I

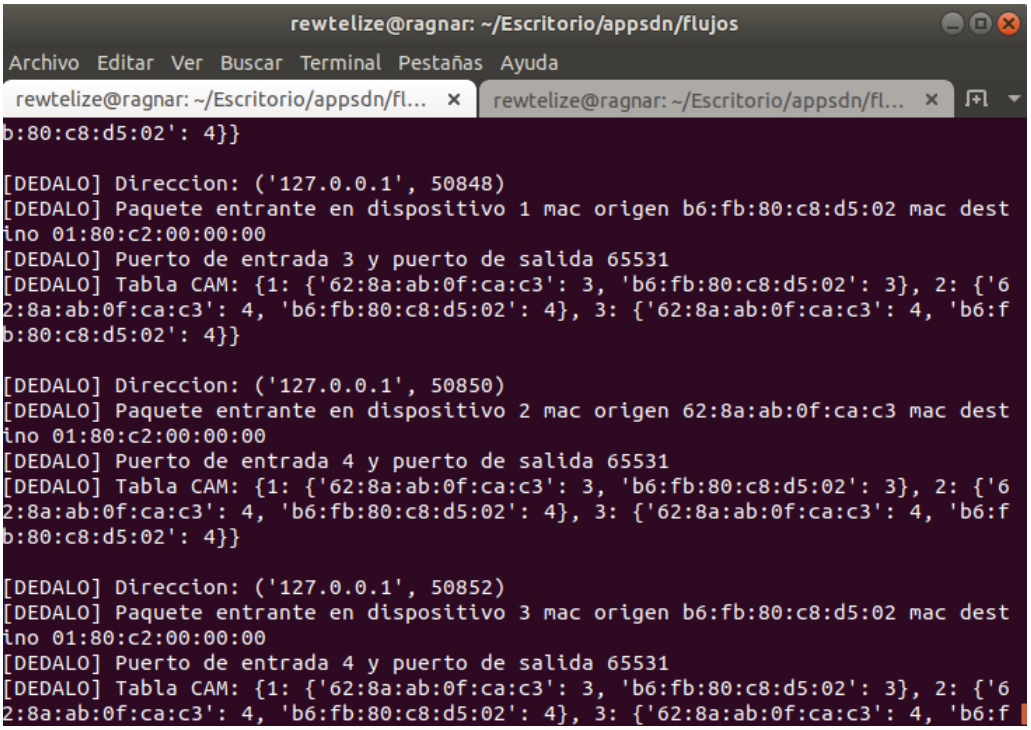

Figura 1.49: Simulación del Prototipo 4 - Parte II

1.10.4.4.4 Entorno real

Siguiendo con la topología del prototipo anterior y, una vez lanzada la aplicación ryu con las políticas implementadas en la consola del servidor, comprobamos que la tabla del controlador contiene toda la información que no cumple la política, que cada conmutador almacena los flujos necesarios y que los `hosts` tienen conectividad con todos los demás a excepción del 192.168.181.14 ya que ha sido conectado en la interfaz 2 de HP-1.

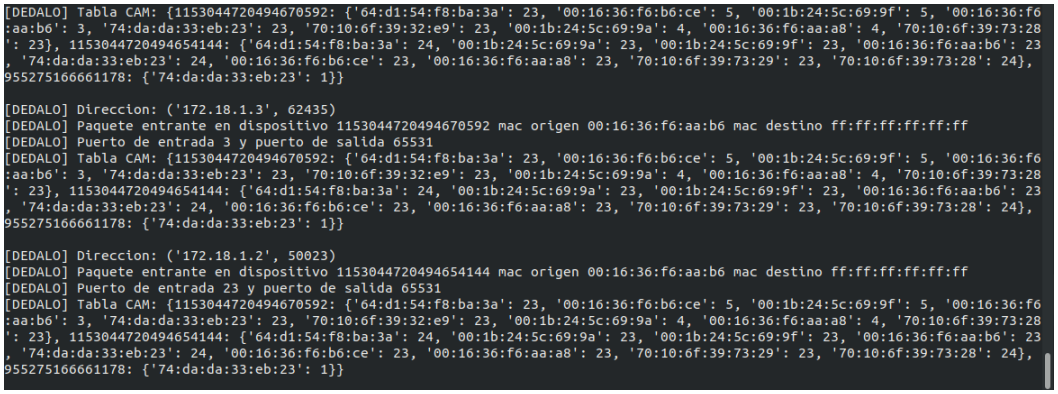

Figura 1.50: Mensajes del controlador del Prototipo 4

```
S1(config)# show openflow instance aggregate flows

 OpenFlow Flow Table

 Flow 1
 Match
  Incoming Port : 24                    Ethernet Type     : Any
  Source MAC    : 74dada-33eb23         Destination MAC   : 001636-f6aab6
  VLAN ID       : Any                   VLAN priority     : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                   IP ToS Bits       : Any
  Source Port   : Any                   Destination Port  : Any
 Attributes
  Priority      : 32768                 Duration          : 729 seconds
  Hard Timeout  : 0 seconds             Idle Timeout      : 0 seconds
  Byte Count    : 362                   Packet Count      : 5
  Controller ID : 1                     Cookie            : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                  : 23

 Flow 2
 Match
  Incoming Port : 23                    Ethernet Type     : Any
  Source MAC    : 001636-f6aab6         Destination MAC   : 74dada-33eb23
  VLAN ID       : Any                   VLAN priority     : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                   IP ToS Bits       : Any
  Source Port   : Any                   Destination Port  : Any
 Attributes
  Priority      : 32768                 Duration          : 729 seconds
  Hard Timeout  : 0 seconds             Idle Timeout      : 0 seconds
  Byte Count    : 362                   Packet Count      : 5
  Controller ID : 1                     Cookie            : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                  : 24
```

Figura 1.51: Flujos de HP-1 del Prototipo 4

```
S2(config)# show openflow instance aggregate flows

 OpenFlow Flow Table

 Flow 1
 Match
  Incoming Port : 23                    Ethernet Type    : Any
  Source MAC    : 74dada-33eb23         Destination MAC  : 001636-f6aab6
  VLAN ID       : Any                   VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                   IP ToS Bits      : Any
  Source Port   : Any                   Destination Port : Any
 Attributes
  Priority      : 32768                 Duration         : 738 seconds
  Hard Timeout  : 0 seconds             Idle Timeout     : 0 seconds
  Byte Count    : 362                   Packet Count     : 5
  Controller ID : 1                     Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                  : 3

 Flow 2
 Match
  Incoming Port : 3                     Ethernet Type    : Any
  Source MAC    : 001636-f6aab6         Destination MAC  : 74dada-33eb23
  VLAN ID       : Any                   VLAN priority    : Any
  Source Protocol Address : Any
  Target Protocol Address : Any
  IP Protocol   : Any                   IP ToS Bits      : Any
  Source Port   : Any                   Destination Port : Any
 Attributes
  Priority      : 32768                 Duration         : 738 seconds
  Hard Timeout  : 0 seconds             Idle Timeout     : 0 seconds
  Byte Count    : 362                   Packet Count     : 5
  Controller ID : 1                     Cookie           : 0x0
  Flow Location : Software
  Hardware Index     : NA
  Reason Code        : 15
  Reason Description : Rule cannot be accelerated in hardware
 Actions
    Output                  : 23
```

Figura 1.52: Flujos de HP-2 del Prototipo 4

```
[admin@MikroTik] > openflow flow print
Flags: I - inactive
 #   SWITCH                  MATCH                   ACTIONS
[admin@MikroTik] >
```

Figura 1.53: Flujos de Mikrotik del Prototipo 4

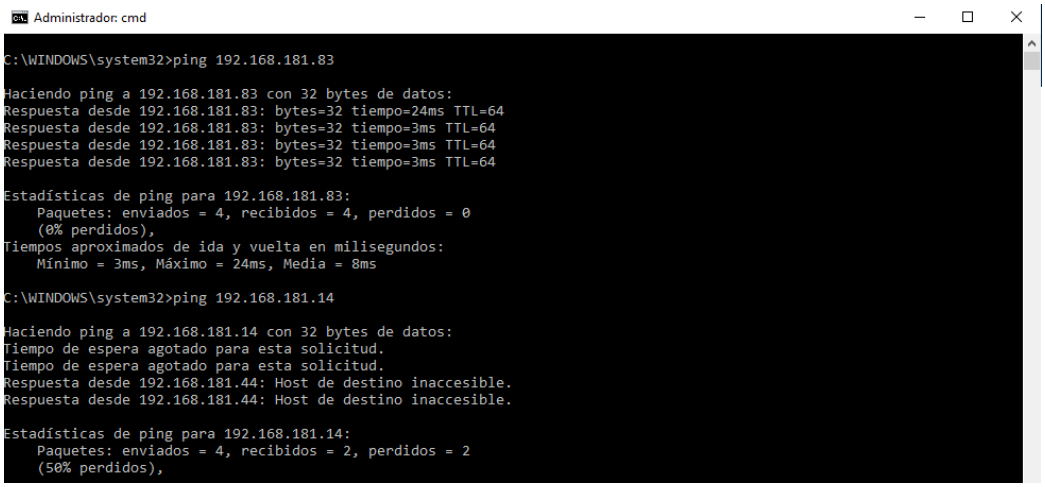

Figura 1.54: Conectividad entre hosts de Mikrotik con HP-1 y HP-2 del Prototipo 4

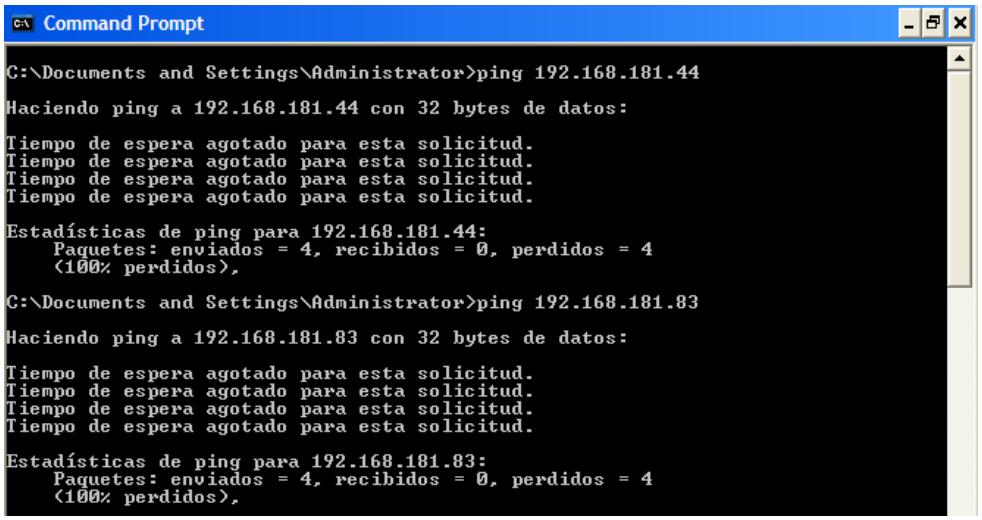

Figura 1.55: Conectividad entre hosts de HP-1 y HP-2 del Prototipo 4

1.10.4.5. Prototipo 5: Implementación rutas de respaldo

1.10.4.5.1 Objetivo

En esta iteración, simularemos hasta tres servidores, donde:

- El Servidor 1 tiene un archivo servidor1.html, copia del index.html de uca.es
- El Servidor 2 tiene un archivo servidor2.html, copia del index.html de stackoverflow.com

- El Servidor 3 recibe ambos y mostrará uno u otro según qué casos:
  - Si tiene conexión con el Servidor 1, mostrará servidor1.html.
  - Si no, mandará un correo al administrador de la red y mostrará servidor2.html.

Para comprobar que el Servidor 3 no tiene conexión con el Servidor 1, tiene que cumplirse que no haya ping entre ambos y que al lanzar un `snmp walk` tampoco se reciban `bytes` en la comunicación.

1.10.4.5.2 Implementación

En este caso, el controlador preguntará a ambos servidores por el código html. Si el primero falla (`uca.es`) entonces mostrará el contenido del segundo `StackOverflow`.

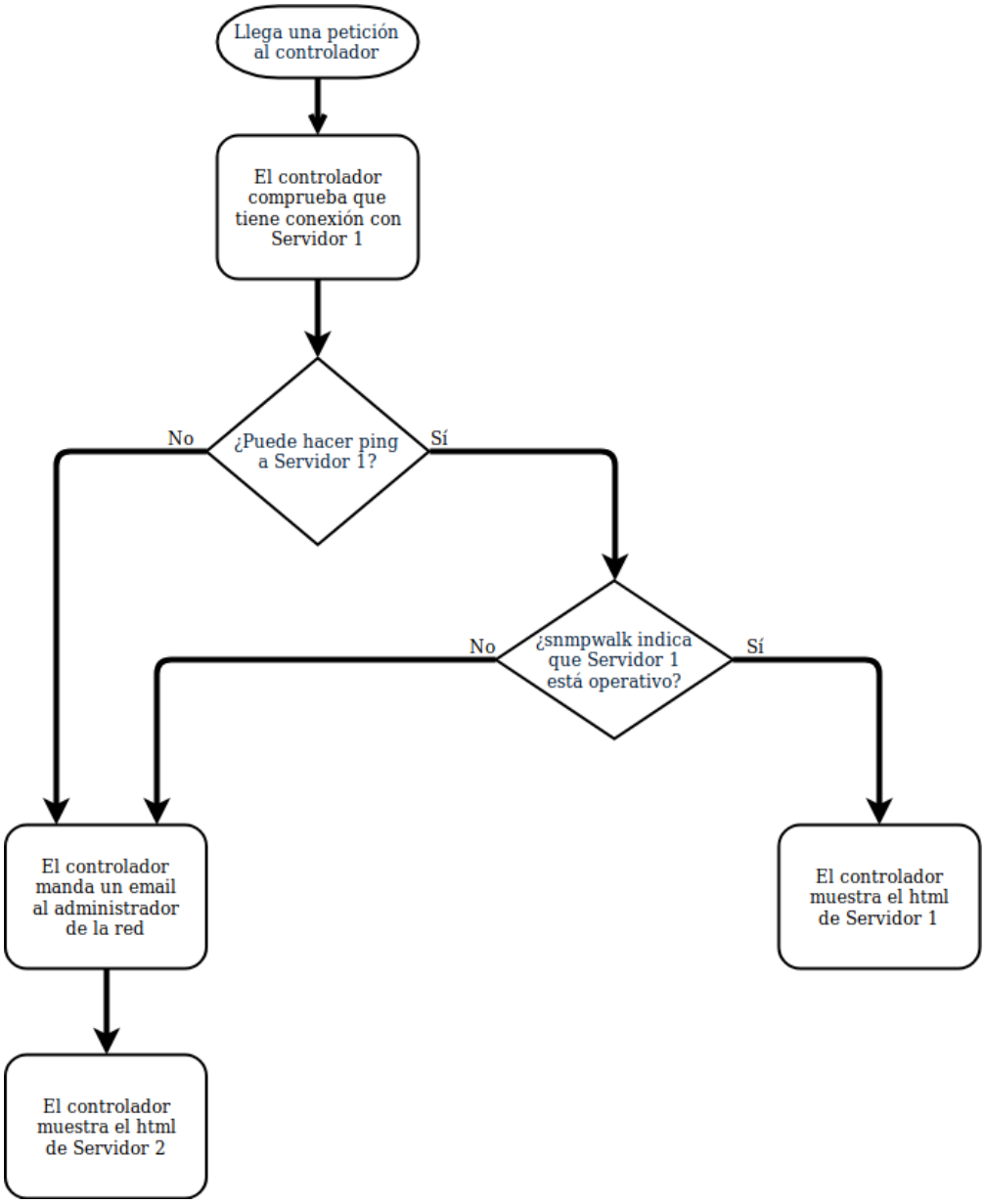

Figura 1.56: Diagrama de flujo del Prototipo 5

```
def mandar_mensaje(request):
    subject = "Servidor no operativo"
    message = "El servidor no responde al protocolo ICMP y el
               contenido en bytes devuelto por un snmpwalk es 0
               \nEl servidor de respaldo ha empezado a sustituirle"
    toemail = "javier.barroso.practicas@dipucadiz.es"
```

```
    email = EmailMessage(subject, message, to=[toemail])
    email.content_subtype= 'html'
    email.send()

def existe_ping():
    response = os.system("ping -c 1 172.18.1.4")

    if response == 0:
      return True
    else:
      return False

def existe_snmpwalk():
    file = open('file.txt', 'w')
    oids = ""
    os.system('snmpwalk -v2c -c public 172.18.1.4
        1.3.6.1.2.1.2.2.1.8.2 > file.txt')
    with open('file.txt', 'r') as f:
        oids = oids + "\n" + str(f.readlines())

    if oids.find("INTEGER: 1") == -1:
        return True

    else:
        return False

def rutas_respaldo(request):
    filename = wget.download('http://www.uca.es')
    os.system('mv download.wget servidor1.html')

    filename = wget.download('https://stackoverflow.com/')
    os.system('mv download.wget servidor2.html')

    if existe_ping() and existe_snmpwalk():
        os.system('mv servidor1.html
            ./RDF/Gestion/Templates/servidor1.html')
        return render(request, 'servidor1.html')
    else:
        os.system('mv servidor2.html
            ./RDF/Gestion/Templates/servidor2.html')
        mandar_mensaje()
        return render(request, 'servidor2.html')
```

1.10.4.5.3 Simulación

Para la misma url `http://127.0.0.1:8000/RutasRespaldo` obtendremos la pantalla de uca.es si la conexión cumple los requisitos previamente mencionados:

Figura 1.57: Simulación Servidor 1

Y, en caso contrario, mostrará la pantalla de StackOverflow:

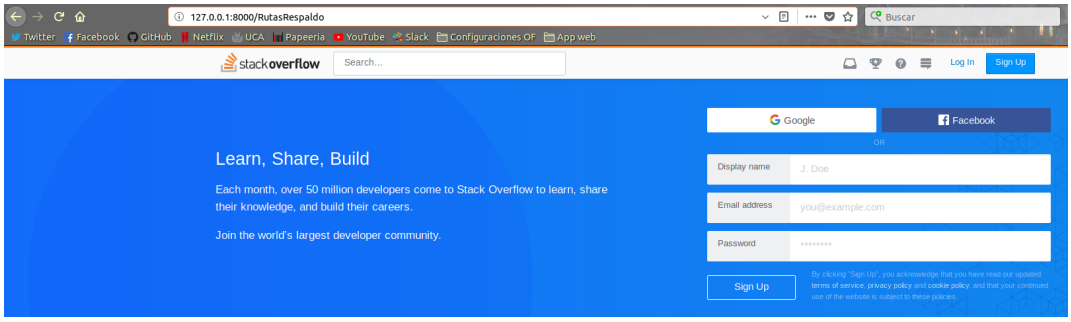

Figura 1.58: Simulación Servidor 2

1.10.4.5.4 Entorno real

Al lanzar la aplicación ryu obtendremos las mismas pantallas que en la simulación, a excepción de la IP con la que abrimos el navegador.

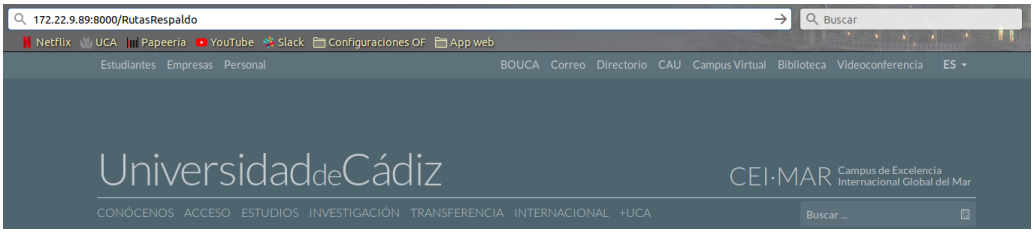

Figura 1.59: Ruta hacia Servidor 1 en entorno real

Y, en caso contrario, mostrará la pantalla de StackOverflow y se mandará un correo:

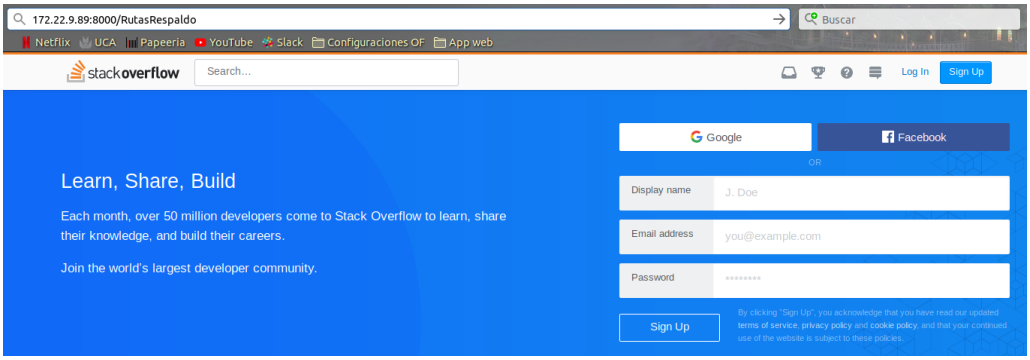

Figura 1.60: Ruta hacia Servidor de respaldo en entorno real

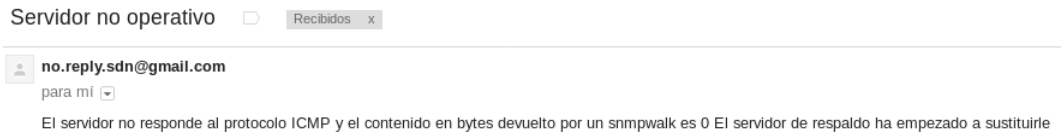

Figura 1.61: Email de servidor de respaldo activo

1.10.4.6. Prototipo 6: Aplicación web SDN

1.10.4.6.1 Objetivo

El último prototipo consistirá en implementar la capa de aplicación de una arquitectura SDN. Como se trata de una aplicación web, implementaremos en un servidor web la interfaz a través del `framework` de desarrollo "Django"; además de un servidor de base de datos con el SGBD MySQL.

Esta aplicación ha recibido el nombre de DÉDALO.

1.10.4.6.2 Topología final

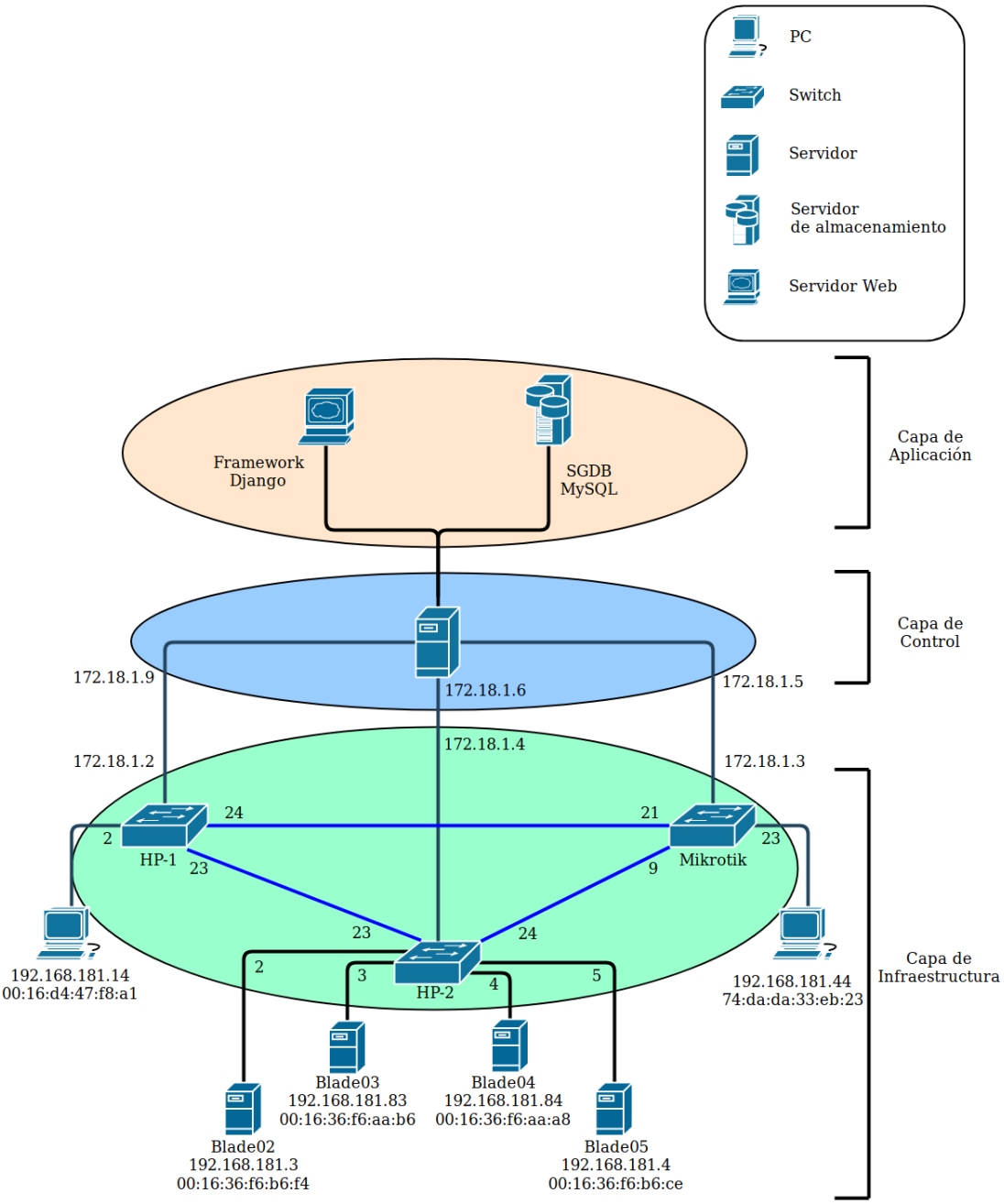

Figura 1.62: Topología final (Prototipo 6)

Esta topología se implementa con el mismo archivo python usado para la topología del prototipo tres, la capa de aplicación está contenida en el propio equipo que

está simulando ser el controlador; es decir, dicho equipo tiene instalado Apache para el despliegue de la web por el protocolo HTTP y el SGBD MySQL para la base de datos.

1.10.4.6.3 Implementación

La implementación de esta aplicación web se ha desarrollado siguiendo las especificaciones del sistema, recogidas en el capítulo 4 Especificaciones del sistema.

En el siguiente enlace se adjunta el repositorio git para que cualquier usuario pueda descargarlo, consultar el código e interactuar con él:

https://github.com/rewtelize/appsdn

1.10.4.6.4 Interfaz

Se adjunta a continuación algunos pantallazos de la aplicación, en concreto, se muestra la pantalla de Administración y la pantalla de gestión de Usuarios.

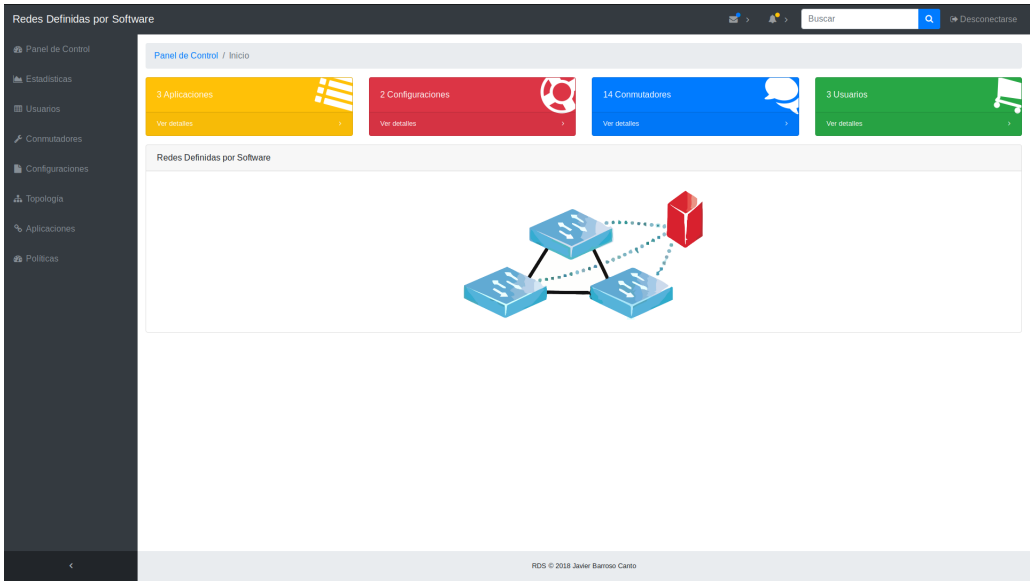

Figura 1.63: Pantalla Administración de DÉDALO

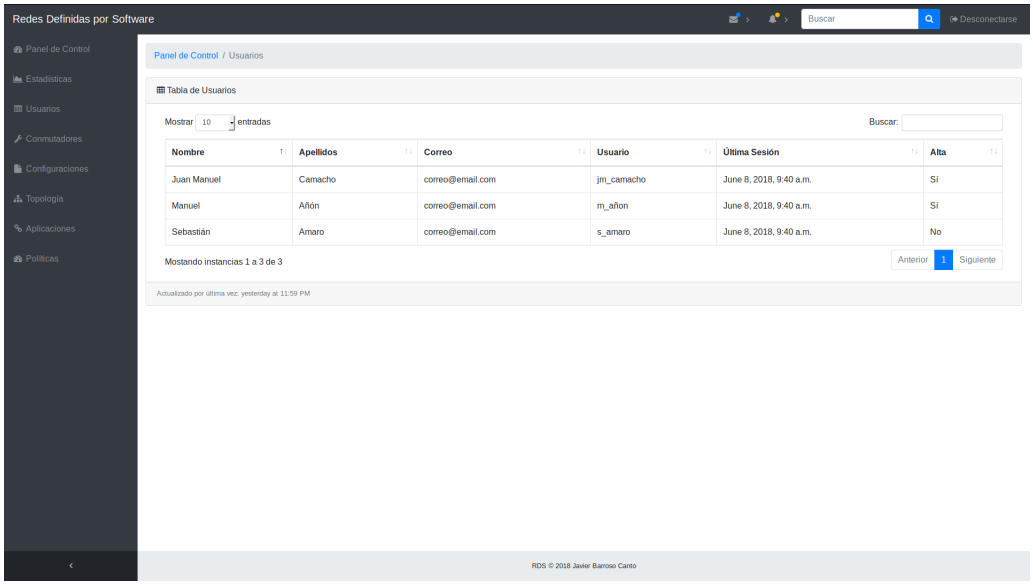

Figura 1.64: Pantalla Usuario de DÉDALO

1.11. Planificación temporal

El plazo de ejecución de este proyecto se llevará a cabo durante cuatro meses, concretamente entre el 1 de Marzo y el 1 de Julio de 2018. Los hitos de la planificación se resumen en los siguientes:

Hito 1. Estudio de SDN

Hito 2. Estudio de las alternativas

Hito 3. Simulación de las alternativas

Hito 4. Reunión en EPICSA

Hito 5. Estudio de la red de EPICSA

Hito 6. Análisis de requisitos

Hito 7. Implementación del controlador de la solución propuesta

Hito 8. Implementación de prototipos

- *a*) Comportamiento de switch
- *b*) Conexiones de varios switches al mismo controlador
- *c*) Implementación de políticas
- *d*) Implementación rutas de respaldo

e) Aplicación web SDN

Hito 9. Documentación

Estos hitos se disponen en la Figura 1.65 mostrando la fecha de cada uno de ellos y una representación gráfica:

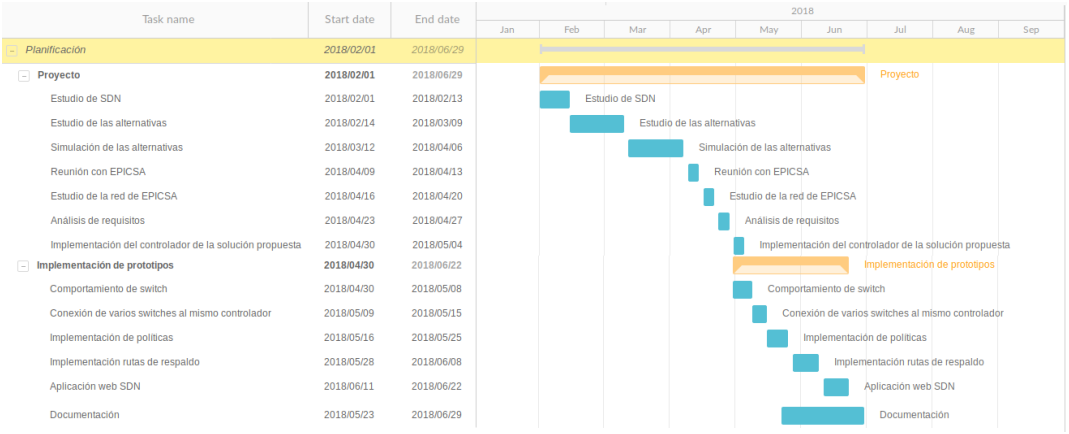

Figura 1.65: Diagrama de Gantt

1.12. Resumen del presupuesto

Se recoge, a continuación, en la Tabla 1.13, el coste total de la ejecución de este proyecto para la empresa que ha de hacerse cargo.

	Precio (€)
Equipamiento	14.970,97
Personal	11.040
Software	0
Total	**26.010,97 €**

Tabla 1.13: Resumen del presupuesto

1.13. Orden de prioridad de los documentos básicos del proyecto

1. Memoria
2. Estudio teórico

3. Anexos
4. Especificaciones del sistema
5. Presupuestos

Estudio de las redes definidas por software (SDN) y desarrollo de un prototipo para la Diputación de Cádiz

Estudio teórico

CLIENTE	EMPRESA PROVINCIAL DE INFORMACIÓN DE CÁDIZ, S.A. (EPICSA)
	PLAZA MADRID S/N, EDIFICIO CARRANZA, FONDO SUR, LOCAL 10, 11010 CÁDIZ
	956 26 15 00
SUMINISTRADOR	JAVIER BARROSO CANTO
	INGENIERO INFORMÁTICO
	49564855-Q
	JAVIER.BARROSOCANTO@ALUM.UCA.ES

FIRMA DEL CLIENTE **FIRMA DEL SUMINISTRADOR**

Cádiz, julio 2018

Estudio teórico

Índice

2.1. Introducción

Las Redes Definidas por Software constituyen un nuevo paradigma de las redes. Se trata de una arquitectura de red emergente donde el plano de control queda desagregado del plano de datos; es decir, el conjunto de mensajes originados por los usuarios (plano de datos) queda desvinculado del transporte de los mismos (plano de control). Para lograrlo, la lógica de control se crea en un controlador. Este mandará determinadas tablas a los distintos dispositivos de red para indicar qué tráfico y cómo debe enviarse.

El hecho de implementar el plano de control en un controlador, facilita la comunicación con aplicaciones de red y el desarrollo de políticas de control mediante las `northbound` APIs; y, mediante las `southbound` APIs, se consigue la comunicación con la infraestructura `hardware`.

Hoy en día, la principal `southbound` API es "OpenFlow", estandarizada por la `Open Networking Foundation` (ONF). Existen otras de menor renombre como "OnePK" de Cisco o "NETCONF" de `Internet Engineering Task Force` (IETF), pero como muchas otras, nacieron con un propósito específico y no como un propósito general de SDN.

Por otro lado, en la actualidad no existe ninguna `northbound` API estandarizada. La ONF señaló que «Normalizar las APIs sofocaría la innovación en el mercado, y una sola API no serviría para todos los casos de uso y todos los ambientes». Por esta razón, existen numerosos controladores con diferentes `northbound` APIs que varían en función de la aplicación final que se pretende desarrollar. [1]

2.2. Tipos de SDN

2.2.1. SDN basado en dispostivos

Los dispositivos del plano de datos son programables a través de las aplicaciones que son ejecutadas en un mismo dispositivo, como en el caso de Cisco OnePK, el cuál permite a los programadores crear aplicaciones que interactúen con otros dispositivos de Cisco. [3]

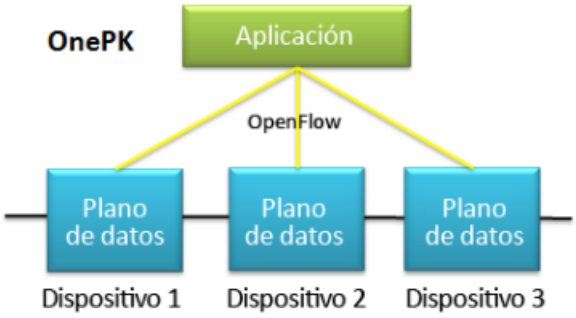

Figura 2.1: SDN basado en dispositivos

2.2.2. SDN basado en controladores

Este tipo de SDN centraliza un servidor el cuál es el encargado de "pensar" por el resto de dispositivos. Es el más implementado actualmente en el mundo de las SDN porque posee varias ventajas sobre el resto de tipos: reduce la complejidad de la red, los cambios de estado entre dispositivos se anuncian más rápidamente, permite automatizar tareas...

Existen varios controladores hoy en día, unos más famosos que otros, que trabajan este tipo de SDN. En este documento se implementa este tipo de SDN y se estudian hasta cuatro opciones distintas para el controlador: OpenDaylight, Ryu, HP VAN SDN y ONOS.

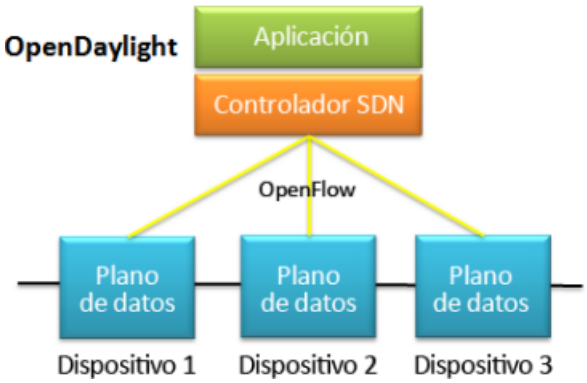

Figura 2.2: SDN basado en controladores

2.2.3. SDN basado en políticas

Este tipo complementa al anterior añadiéndo una políticas incluyendo una capa de políticas adicional para dar un nivel de abstracción mayor. Utiliza las aplicaciones

integradas que automatizan tareas de configuración avanzadas mediante un flujo de trabajo guiado y una GUI fácil de usar que no requiere habilidades de programación. [3]

Figura 2.3: SDN basado en políticas

2.3. Arquitectura

Como se ha mencionado en la introducción, la diferencia entre una red convencional y otra SDN es que, en la última, existen dos planos que ya no se implementan juntos: datos y control. No obstante, si atendemos a las distinas imágenes sobre la arquitectura, muchos investigadores la representan en tres capas claramente diferenciadas: aplicación, control e infraestructura; tal y como se muestra en la Figura 2.4 [5]

- **Capa de aplicación**: Aquí se encontrarían ubicadas las aplicaciones SDN. Estas aplicaciones comunican, ya sea explícitamente o de forma automatizada, sus requisitos de red, así como el comportamiendo deseado de la misma, al controlador SDN a través de la `northbound` API.
- **Capa de control**: Un controlador SDN es una entidad centralizada que se encarga de que los dispositivos del plano de datos tengan un comportamiento adecuado como para satisfacer las peticiones del plano de aplicación. Tal y como se aprecia en la figura antes mencionada, el plano de control hace uso de las `southbound` APIs para la comunicación con la infraestructura `hardware`.
- **Capa de infraestructura**: Este plano comprende un conjunto de dispositivos de red, virtuales o físicos, que delegan su inteligencia al controlador y pasan a ser simples unidades de conmutación de tráfico. [1]

Esta representación en forma de capas se relaciona con la definición de SDN de la siguiente manera:

- **Plano de control**: Compuesta por la capa de aplicación y la capa de control.
- **Plano de datos**: Conformada por la capa de infraestructura.

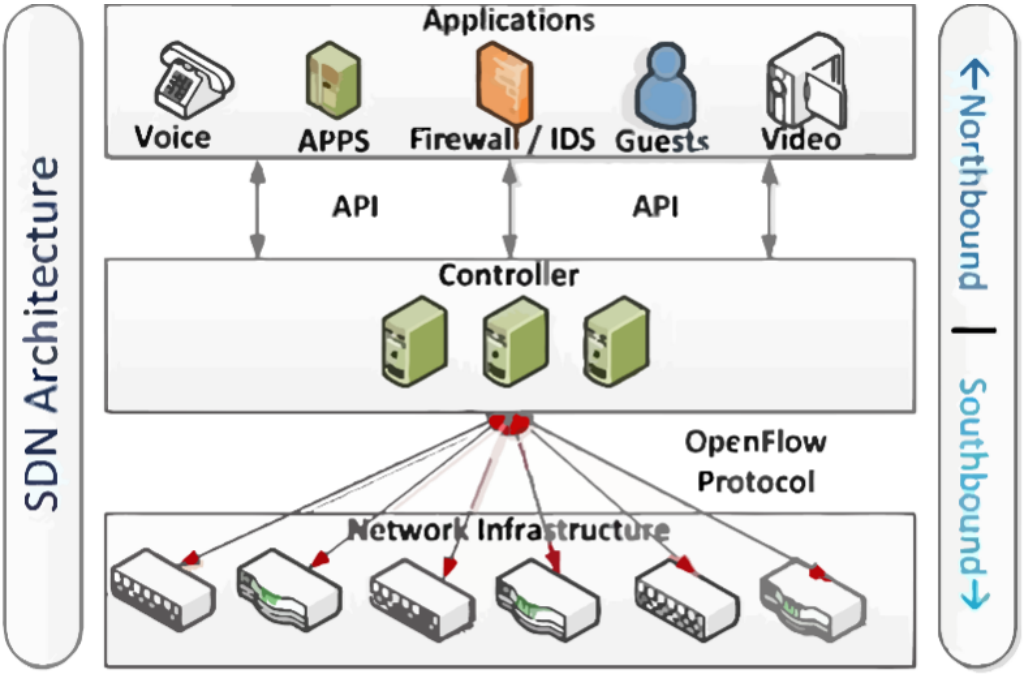

Figura 2.4: Arquitectura de una SDN

Para terminar de explicar la arquitectura, tenemos que mencionar las dos APIs de las que hace uso el controlador para comunicarse con los planos de aplicación y datos:

- **Northbound API**: Estas interfaces de programación son usadas para la comunicación con el controlador y las aplicaciones que estána actuando sobre una red. Esta API facilita la innovación y automatiza determinadas accciones para conseguir que la red cubra, mediante programación, las diferentes necesidades que un servicio pudiera necesitar. [28]
- **Southbound API**: Estas APIs, por el contrario, comunican al controlador con los diferentes dispositivos de la red (conmutadores, enrutadores...). De esta forma, al centralizar todas las decisiones de dichos dispositvos sobre un controlador, facilitamos un control eficiente sobre la red y permitimos que dicho controlador sea capaz de actuar en tiempo real a las distintas demandas o necesidades que la red pudiera acusar. [29]

2.4. Plano de Control

2.4.1. Capa de Aplicación

Las aplicaciones SDN son programas que, ya sea de manera explícita o implícita, comunique de forma programada los requisitos de la red y el comportamiento deseado de la misma al controlador SDN a través de una Northbound API. Adicionalmente, pueden mostrar una vista abstracta de la red para una posible toma de decisiones.

Comúnmente, una aplicación SDN consiste en una aplicación lógica y una o más Northbound APIs.

2.4.2. Capa de Control

2.4.2.1. El controlador

El controlador SDN es el encargado de definir los flujos de datos y es el equivalente al sistema operativo de la red, ya que de él dependen las comunicaciones entre las aplicaciones y los dispositivos, es decir, el controlador es el encargado de traducir las peticiones de la capa superior a los de la capa inferior.

Por ello, el controlador es el que dicta e imbuye el comportamiento de la red en general, a partir de las peticiones de las aplicaciones.

2.4.2.2. Centralización

La centralización de los servicios, que fue considerado una deficiencia por temas de escalabilidad, actualmente es considerada una ventaja: hoy en día se divide la red en subconjuntos o zonas de tamaño reducido como para tener una estrategia de control común; consiguiendo de esta manera que los cambios de estado en los dispositivos se propaguen más rápidos que con un sistema distribuído.

Frente a fallos de los controladores centralizados, se pueden utilizar controladores que tomen el control si se presentan problemas en el controlador principal; y que todo siguiera resultando de manera transparente al plano de datos.

No obstante, en la arquitectura de las SDN debe existir un equilibrio entre la complejidad de la gestión y el control centralizado; ya que si aumentamos la cantidad de controladores resultará más complejo la gestión de la red.

2.4.2.3. Programabilidad

Al implentar la centralización del plano de control, se facilita que se puedan desarrollar políticas de control, simplemente cambiando el `software`; consiguiendo, a partir de las `Northbound APIs` que su gestión sea de forma dinámica de acuerdo al estado de la red.

Este plano de control programable es el aspecto más importante de las SDN, pues consigue que la red se pueda dividir en distintas redes virtuales con políticas diferentes y residan en la misma infraestructura `hardware`.

2.4.2.4. Control basado en flujos

El control basado en flujos reduce significativamente el tráfico entre el controlador y los dispositivos del plano de datos, ya que si se toma una decisión de control para un paquete del flujo, se puede reutilizar la misma decisión para los restantes paquetes.

Un flujo se puede identificar por cabeceras en los paquetes o el puerto de entrada por el que lo recibió. [1]

2.5. Plano de Datos

Un conmutador OpenFlow puede poseer una o varias tablas de flujo y una acción asociada a cada entrada de dicha tabla; y un canal OpenFlow con el que comunicarse con el controlador e intercambiar comandos y paquetes a partir del protocolo OpenFlow.

Según la marca, los campos pueden variar, pero los principales de un flujo son:

- **Match Fields**: utilizado para la detección de paquetes. Almacena el puerto de entrada y la cabecera de los paquetes.
- **Priority**: define la prioridad del paquete y, junto al campo Match Fields identifican los paquetes.
- **Counters**: contador que se actualiza por cada paquete identificado.
- **Instructions**: Modifica el conjunto de acciones.
- **Timeouts**: Da valor al tiempo que puede permanecer como máximo una entrada en la tabla.

- **Cookie**: Datos elegidos por el controlador y utilizados por este para filtrar estadísticas, modificaciones...

Se muestra, a modo de ejemplo, un flujo albergado en una tabla de flujos de un conmutador HP 2920:

```
OpenFlow Flow Table

Flow 1
Match
 Incoming Port : 1                    Ethernet Type     : Any
 Source MAC    : 0016d4-47f8a1        Destination MAC   : 001636-f6b6ce
 VLAN ID       : Any                  VLAN priority     : Any
 Source Protocol Address : Any
 Target Protocol Address : Any
 IP Protocol   : Any                  IP ToS Bits       : Any
 Source Port   : Any                  Destination Port  : Any
Attributes
 Priority      : 32768                Duration          : 360966 seconds
 Hard Timeout  : 0 seconds            Idle Timeout      : 0 seconds
 Byte Count    : 3638                 Packet Count      : 47
 Controller ID : 1                    Cookie            : 0x0
 Flow Location : Software
 Hardware Index     : NA
 Reason Code        : 15
 Reason Description : Rule cannot be accelerated in hardware
Actions
   Output                : 21
```

Figura 2.5: Ejemplo de un flujo

2.6. Northbound APIs

Las Northbound APIs son probablemente las APIs más críticas en el entorno SDN, ya que el valor de este está vinculado a nuevas aplicaciones innovadoras o emergentes que potencialmente puedan admitir/habilitar estas APIs o no. Por este motivo, las Northbound APIs deben diseñarse con el objetivo de admitir una amplia variedad de aplicaciones.

Algunas de las características de las aplicaciones de red que pueden optimizarse con las Northbound APIs son el balanceamiento de carga, `firewalls` u otros servicios de seguridad definidos por `software` y aplicaciones de orquestación a través de recursos en la nube.

Las Northbound APIs también pueden usarse para abstraer el funcionamiento de la red, de modo que los desarrolladores de aplicaciones puedan hacer cambios en ella para adaptarse a las necesidades de la aplicación que se estuviera desarrollando sin tener que entender exactamente qué significa eso para la red. [28]

2.7. Southbound APIs

2.7.1. OnePK

OnePK es una Southbound API, propietaria de Cisco, formada por un conjunto de herramientas que permite a los programadores desarrollar aplicaciones que puedan integrarse fácilmente en un entorno Cisco.

Cisco onePK está diseñado para ayudar a los usuarios a obtener acceso a la información almacenada dentro de su red y tener un control más directo sobre los flujos y rutas de la red. OnePK permite a los usuarios personalizar y programar la red, de modo que se pueda adaptar más fácilmente a las necesidades de negocios y aplicaciones que cambian rápidamente. [30]

La arquitectura onePK de Cisco se compone de tres elementos principales:

- Capa de presentación: alberga las bibliotecas de API que los programadores pueden usar dentro de sus propias aplicaciones.
- Infraestructura API: proporciona acceso a funciones internas en enrutadores o conmutadores.
- Canal de comunicación: Proporciona un canal extensible, seguro y rápido entre la aplicación y el elemento de red.

Por otro lado, onePK se puede implementar de tres formas distintas:

1. `Hosting` de procesos: el contenedor Linux aloja la aplicación en la red misma.
2. `Blade Hosting`: la aplicación se aloja en un blade de hardware específico dentro del mismo chasis (o puerto) que el elemento de red.
3. Alojamiento de nodo final: un dispositivo de usuario final aloja la aplicación.

Se presenta, en la siguiente figura [30], una representación gráfica:

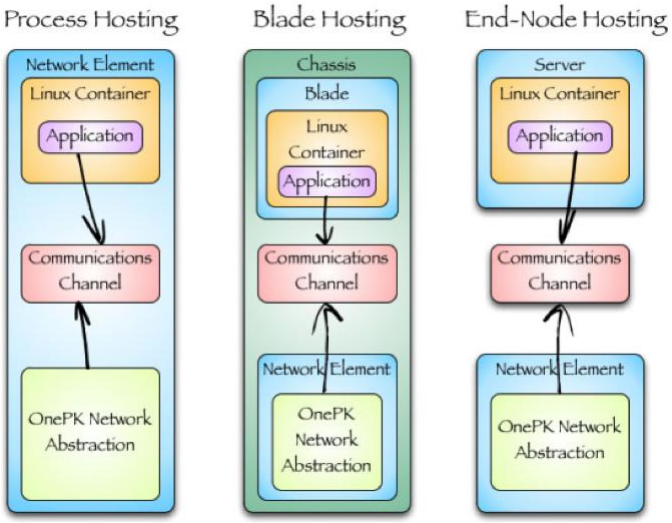

Figura 2.6: Opciones de desarrollo onePK

2.7.2. OpenFlow

OpenFlow nació en 2007 en la universidad de Stanford, Palo Alto (California); a partir del proyecto "Ethane". Este proyecto fue desarrollado por un alumno de doctorado llamado Martín Casado y la empresa HP. La idea de "Ethane" era separar una parte de la red para hacer determinados experimentos sin que el resto de usuarios finales no se vieran afectados. En ese momento, ya se estaban tratando dos puntos que formarían parte, más adelante, de las arquitecturas SDN: un flujo (compuesto por dos nodos y un tráfico) y un controlador que lo gestionara. En Noviembre de ese mismo año, nace OpenFlow. [22]

Este protocolo es la primera interfaz estandarizada definida entre las capas de control y datos de una arquitectura SDN. OpenFlow permite el acceso directo, así como la manipulación del plano de datos de los dispositivos de red.

Esta southbound API se implementa en ambas capas, tanto en el controlador como en los diferentes equipos que conforman la red. OpenFlow utiliza el concepto de "flujo" para identificar el tráfico de red basado en las relaciones programadas en el controlador SDN. Todo ello permite al administrador de red definir reglas basadas en determinados parámetros tales como patrones, aplicaciones, necesidades de la nube... [2]

En la Figura 2.7 [9] observamos el diagrama que rige el comportamiento del protocolo con la entrada de un flujo. Tal y como se describe, cuando un paquete llega a un conmutador, lo primero que hace este es comprobar si está emparejado a una tabla. Si es así, típicamente el conmutador ejecutará una serie de instrucciones

o bien lo redirigirá a otra tabla, con una referencia previamente definida, para que ella se encargue de su tratamiento.

Si no existe ningun emparejamiento para el paquete entrante, se manda al controlador. Este podrá descartarlo o crear un nuevo flujo, mediante la creación de una nueva entrada en alguna de las tablas.

En conclusión, un conmutador OpenFlow, dependiendo de si un paquete entrante está emparejado a una tabla o no, puede realizar las siguientes acciones básicas:

- Reenviar un paquete.
- Descartar un paquete.
- Encapsular y reenviar un paquete al controlador.

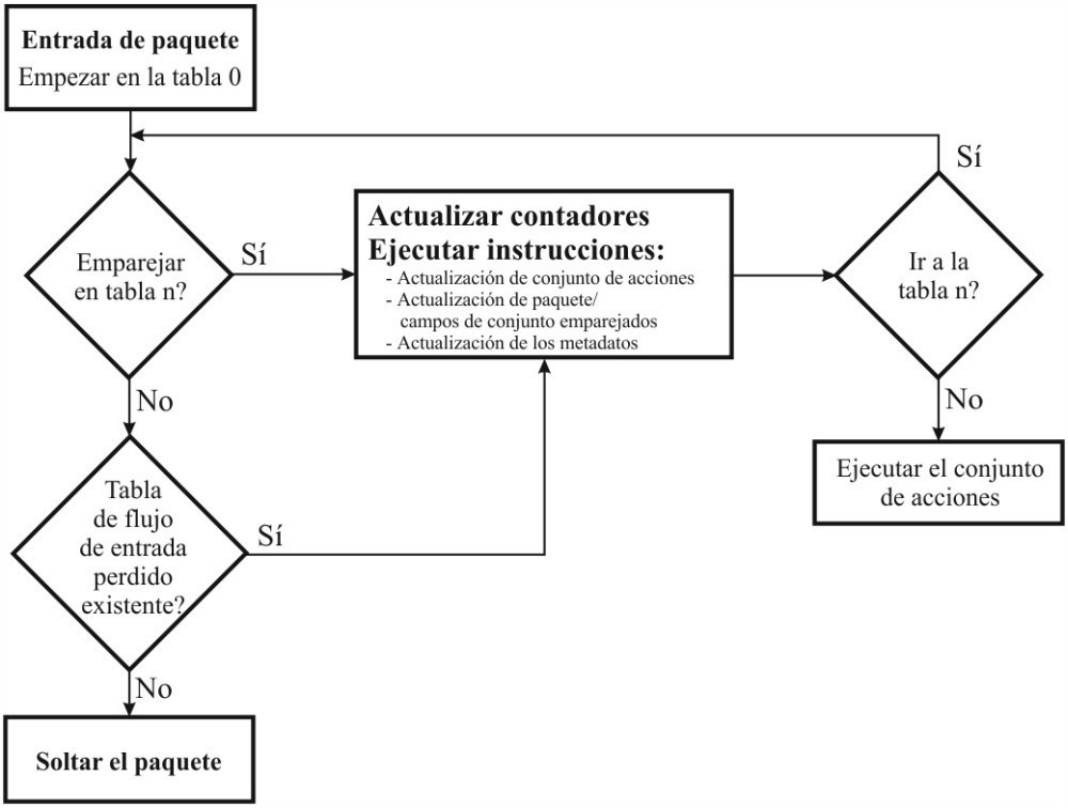

Figura 2.7: Diagrama de flujo del protocolo OpenFlow

Estudio de las redes definidas por software (SDN) y desarrollo de un prototipo para la Diputación de Cádiz

Anexos

CLIENTE	EMPRESA PROVINCIAL DE INFORMACIÓN DE CÁDIZ, S.A. (EPICSA)
	PLAZA MADRID S/N, EDIFICIO CARRANZA, FONDO SUR, LOCAL 10, 11010 CÁDIZ
	956 26 15 00
SUMINISTRADOR	JAVIER BARROSO CANTO
	INGENIERO INFORMÁTICO
	49564855-Q
	JAVIER.BARROSOCANTO@ALUM.UCA.ES

FIRMA DEL CLIENTE **FIRMA DEL SUMINISTRADOR**

Cádiz, julio 2018

Anexos

Índice

3.1. Documentación de partida

A fecha 2 de Abril de 2018, se propone una reunión con Manuel Añón Rodríguez con el objetivo de recabar información sobre determinados puntos:

- Estado de la situación actual de las redes en EPICSA.
- Activos a tener en cuenta durante la ejecución e implantación de la aplicación.
- Datos a almacenar y mostrar en la interfaz.
- Funcionalidades demandadas.
- Estado objetivo de la red.

Se recoge, a continuación, la entrevista con el mencionado coordinador del departamento de redes y telecomunicaciones tras las presentaciones pertinentes.

P: Sería interesante que empezaramos hablando sobre la topología de la red corportativa.

R: En EPICSA administramos la red corporativa de la Diputación Provincial de Cádiz, que interconecta todas las sedes provinciales de la Diputación y todos los ayuntamientos de la Provincia de Cádiz con menos de 20.000 habitantes. Pondremos a su disposición, documentación técnica que Telefónica desarrolló durante la realización del proyecto de construcción de nuestra red corporativa.

P: ¿Qué funcionalidades creen que debe tener la aplicación?

R: Ahora mismo, creemos que tener un aplicación capaz de desviar paquetes en un sistema de respaldo es una de las mejores ventajas de las redes SDN. Nos gustaría que, si el enlace que va desde el conmutador hasta nuestro centro de respaldo está caído, el sistema sea capaz, de manera automatizada, de desviar los paquetes hacia otro servidor de respaldo de `backup`.

P: ¿Qué dispositivos tienen actualmente en su CPD capaces de tratar con el protocolo OpenFlow? ¿Qué esperan de la aplicación SDN?

R: Actualmente contamos con dos conmutadores capaces de trabajar con el protocolo OpenFlow, en concreto dos HP 2920-24G Switch J9726A y un conmutador MikroTik CRS125-24G-1S-IN.

Además, nos gustaría ver cómo se interconectan entre ellos en una simulación de nuestra red del área metropolitana, ya con la separación del plano de control del

plano de datos, a una escala un poco más reducida; con el objetivo de tener una base de conocimiento sobre la cual basar la toma de decisiones en cuanto a la implementación real de SDN de nuestra red en un futuro.

P: ¿Qué otros equipos tendré a mi disposición para poder simular la red definida por software?

R: Podrá usar un servidor HP ProLiant DL380 G5 a modo de controlador, el cuál estará conectado con los tres conmutadores mencionados previamente y dónde incurrirá el plano de control. Además dispondrá de un Blade Fujitsu para contar con dispositivos finales. En todos los dispositivos deberá instalar un sistema operativo de código abierto.

P: ¿Cuántos usuarios tendrán acceso a la aplicación?

R: En principio, hasta tres personas conformamos el departamento de redes en EPICSA, Juan Manuel Camacho, Sebastián Amaro y un servidor; pero queremos contar con la opción de añadir más usuarios, así como de eliminarlos, por si en un futuro fuera necesario.

P: ¿Alguna otra consideración a tener en cuenta en el desarrollo de la aplicación?

R: Es importante que la aplicación sea de calidad, es decir: debe ofrecer cierta seguridad para que no todos los usuarios puedan darse de alta, debe adecuarse funcionalmente a los requisitos que demandamos, la interfaz ha de ser sencilla y visualmente cómoda y debe ser capaz de trabajar en multiplataformas (en cuanto a marcas de conmutadores).

3.2. Mininet

Mininet es un un emulador de red. Ejecuta una colección de `hosts` finales, conmutadores, enrutadores y enlaces en un único `kernel` de Linux. Utiliza una virtualización liviana para hacer que un solo sistema se vea como una red completa, ejecutando el mismo kernel, sistema y código de usuario.

3.2.1. Instalación

Para instalar Mininet basta con descargárnoslo de GitHub y ejecutar su script:
$ git clone git://github.com/mininet/mininet
$ bash mininet/util/install.sh -a

Una vez instalado tenemos hasta tres maneras distintas, detalladas en la siguiente sección, de crear una red para emular.

3.2.2. Simulación por línea de comandos

Para simular una red por línea de comandos es necesario conocer las siguientes cláusulas:

- **controller=remote,ip=<`dirección_ip_controlador`>** crea un controlador manualmente, con la IP del equipo que actuará como controlador.
- **topo** crea la topología. Tenemos dos opciones:
  - **single,<`numero_hosts`>** indica al controlador que se trata de una red compuesta por un conmutador con "n" `hosts` asociados al mismo.
  - **tree,<`profundidad_árbol`>** indica que se trata de un árbol de conmutadores de profundidad "n" y, a los del último nivel, les asocia dos `hosts` a cada uno.
- **mac** asocia direcciones físicas fáciles de recordar e intuitivas.
- **switch ovsk,protocols=OpenFlow<`versión`>** le indica al controlador que va a trabajar con una versión determinada del protocolo OpenFlow.
- **nat** conecta la red simulada con Internet mediante el mecanismo "Network Address Translation" (NAT).

Así, pues, con la siguiente orden, estaríamos simulando una red con un controlador en nuestra máquina local, conectado a un conmutador con tres `hosts`:

$ sudo mn –controller=remote,ip=127.0.0.1 –topo single,3 –mac –switch ovsk,protocols=OpenFlow10

3.2.3. Simulación mediante python

Para emular una red a partir de un archivo python, basta con ejecutar la orden:
$ sudo python topologia.py

Se adjunta, a continuación, un ejemplo de topología con los comandos ya comentados; en este caso, se trata de un controlador conectado a dos conmutadores y estos a un host cada uno.

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.node import OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
from time import sleep

def miRedSimulada():
    info('*** Crea la red net\n')
    net = Mininet(topo=None,
                  build=False,
                  ipBase='127.0.0.1/8')

    info('*** Insertar un controlador en la red\n')
    c0 = net.addController(name='c0',
                           controller=RemoteController,
                           ip='127.0.0.1',
                           protocol='tcp',
                           port=6633)

    info('*** Insertar switches a la red\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)

    info('*** Insertar hosts a la red con una ip dada\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1',
        defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2',
        defaultRoute=None)
```

```
    info('*** Insertar enlaces entre dispositivos\n')
    net.addLink(h1, s1)
    net.addLink(h2, s2)
    net.addLink(s1, s2)

    info('*** Lanza la red net\n')
    net.build()


    info('*** Lanzando controladores\n')
    for controller in net.controllers:
        controller.start()

    info('*** Lanzando switches\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])

    info('*** Pruebas de configuracion y CLI\n')
    net.pingAll()
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    miRedSimulada()
```

3.2.4. Simulación mediante el script "Miniedit"

Este script es otro archivo python que incorpora una interfaz para que sea más sencillo el crear la topología. Para lanzar el script, hay que usar el siguiente comando:
$ sudo python mininet/examples/miniedit.py

Y si todo ha ido bien, visualizaríamos una pantalla como la de la Figura 3.1. Entonces seremos capaz, mediante el arrastre de `hosts` y `switches` crear un nuevo archivo python que, como el punto anterior, basta con ser ejecutado para que Mininet emule la red.

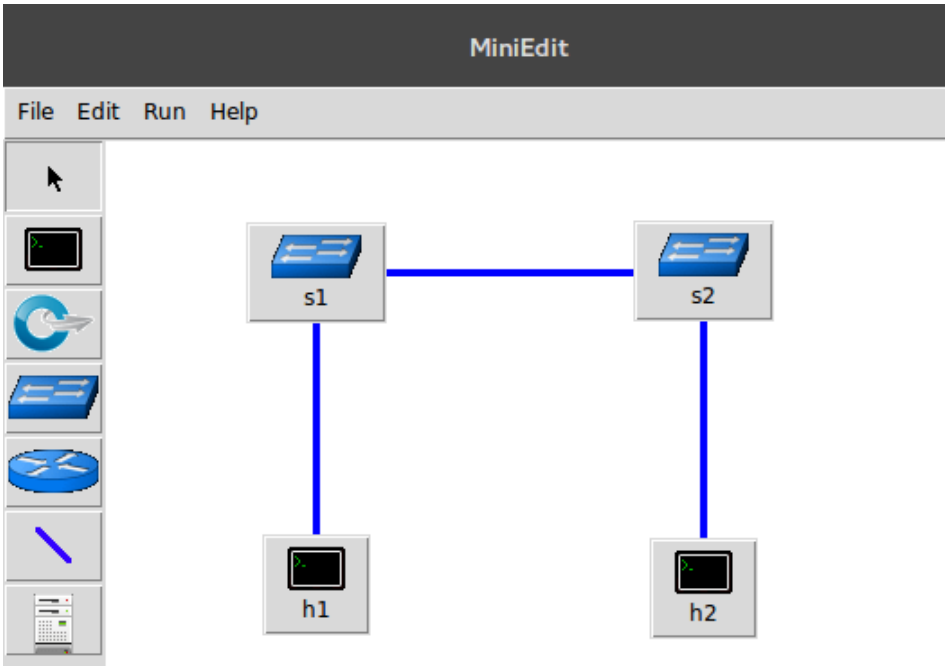

Figura 3.1: Script Miniedit para la simulación de redes en Mininet

3.3. Configuraciones de los conmutadores

3.3.1. Comandos básicos empleados

3.3.1.1. Comandos de HP

3.3.1.2. openflow

Entra en el modo configuración del contexto OpenFlow.

3.3.1.3. openflow instance `instance-name`

Entra en el modo configuración de la instancia OpenFlow.

3.3.1.4. openflow [enable | disable]

Habilita o deshabilita el protocolo.

3.3.1.5. openflow instance [`instance-name` | aggregate] { enable | disable }

Habilita o deshabilita la instancia OpenFlow o el modo "aggregate". Este último, habilita al conmutador para que una sola instancia de OpenFlow gestione todas las VLANs.

3.3.1.6. openflow instance `instance-name` mode { active | passive }

Provoca que la instancia pase a modo:

- Activo: los nuevos flujos que lleguen al conmutador y este no sepa qué hacer con ellos los mandará al controlador.
- Pasivo: Aquellos flujos que el conmutador no sepa tratar no serán mandados al controlador.

3.3.1.7. openflow instance { `instance-name` | aggregate } listen-port [tcp-port] [oobm]

Indica al conmutador a qué puerto del controlador debe escuchar, por defecto es el 6633. El comando da opción de indicarle un nuevo puerto, o bien establecerle el puerto usado para la gestión fuera de banda (oobm).

3.3.1.8. openflow controller-id `ID` ip `ip-address` [port `tcp-port`] controller-interface { vlan vlan-id | oobm }

Establece el controlador de OpenFlow a través de un puerto o una vlan.

3.3.1.9. show openflow instance [`instance-name` | aggregate]

Muestra la información sobre una instancia o del modo “aggregate”.

3.3.1.10. show openflow controllers

Muestra la información sobre los controladores configurados.

3.3.1.11. show openflow instance { `instance-name` | aggregate } flows

Muestra la información sobre los flujos generados en el conmutador a través de la configuración del controlador.

3.3.1.12. Comandos de MikroTik

3.3.1.13. openflow add name=`instance-name` controllers=`ip-address`

Añade una instancia al conmutador a partir de un nombre y una IP con la que conectar con el controlador.

3.3.1.14. openflow port add switch=`instance-name` interface=`tcp-port`

Añade un puerto a una instancia previamente creada.

3.3.1.15. openflow export

Muestra la configuración de openflow en el conmutador, inlcuyendo instancias y puertos.

3.3.1.16. openflow flow print

Muestra información sobre los flujos que se han generado en el conmutador a partir de la configuración del controlador.

3.3.2. Configuración de los conmutadores

3.3.2.1. Conmutador HP-1

```
hostname "S1"
module 1 type j9726a
gvrp
ip routing
snmp-server community "Epics@" restricted
openflow
   controller-id 1 ip 172.18.1.9 controller-interface vlan 50
   egress-only-ports
   instance "ryu"
      listen-port
      member vlan 3
      controller-id 1
      connection-interruption-mode fail-standalone
      enable
      exit
   ip-control-table-mode
   enable
   exit
oobm
   ip address dhcp-bootp
   exit
vlan 1
   name "DEFAULT_VLAN"
   no untagged 1-24
   untagged A1-A2,B1-B2
   no ip address
   exit
vlan 3
   name "DATOS"
   untagged 1-12,21-24
   ip address 192.168.181.80 255.255.255.0
   exit
vlan 50
   name "CONTROL"
   untagged 13-20
   ip address 172.18.1.2 255.255.255.0
   forbid 21-24
   exit

primary-vlan 50
```

```
no tftp server
no autorun
no dhcp config-file-update
no dhcp image-file-update
password manager
password operator
```

3.3.2.2. Conmutador HP-2

```
hostname "S2"
module 1 type j9726a
gvrp
ip routing
snmp-server community "Epics@" restricted
openflow
   controller-id 1 ip 172.18.1.5 controller-interface vlan 50
   egress-only-ports
   instance "ryu"
      listen-port
      member vlan 3
      controller-id 1
      enable
      exit
   ip-control-table-mode
   enable
   exit
oobm
   ip address dhcp-bootp
   exit
vlan 1
   name "DEFAULT_VLAN"
   no untagged 1-24
   untagged A1-A2,B1-B2
   no ip address
   exit
vlan 3
   name "DATOS"
   untagged 1-12,21-24
   ip address 192.168.181.1 255.255.255.0
   exit
vlan 50
   name "CONTROL"
   untagged 13-20
   ip address 172.18.1.3 255.255.255.0
   forbid 21-24
   exit

primary-vlan 50
no tftp server
no autorun
no dhcp config-file-update
```

```
no dhcp image-file-update
password operator
```

3.3.2.3. Conmutador MikroTik

```
# jan/09/2002 03:11:24 by RouterOS 6.42.3
# software id = LF58-9VFF
#
# model = CRS125-24G-1S
/interface bridge
add admin-mac=64:D1:54:F8:BA:3A auto-mac=no comment=\
    "created from master port" name=bridge1 protocol-mode=none
/interface ethernet
set [ find default-name=ether1 ] name=ether1-master
/interface wireless security-profiles
set [ find default=yes ] supplicant-identity=MikroTik
/openflow
add controllers=172.18.1.6 datapath-id=3/64:D1:54:F8:BA:3A disabled=no name=\
    ofswitch1
/snmp community
set [ find default=yes ] authentication-password=1234567890 \
    encryption-password=1234567890
/tool user-manager customer
set admin access=\
    own-routers,own-users,own-profiles,own-limits,config-payment-gw
/interface bridge port
add bridge=bridge1 interface=ether2 priority=0x60
add bridge=bridge1 interface=ether3
add bridge=bridge1 interface=ether4
add bridge=bridge1 interface=ether5
add bridge=bridge1 interface=ether6
add bridge=bridge1 interface=ether7
add bridge=bridge1 interface=ether8
add bridge=bridge1 interface=ether9 priority=0x70
add bridge=bridge1 interface=ether10
add bridge=bridge1 interface=ether11
add bridge=bridge1 interface=ether12
add bridge=bridge1 interface=ether13
add bridge=bridge1 interface=ether14
add bridge=bridge1 interface=ether15
add bridge=bridge1 interface=ether16
add bridge=bridge1 interface=ether17
add bridge=bridge1 interface=ether18
add bridge=bridge1 interface=ether19
add bridge=bridge1 interface=ether20
add bridge=bridge1 interface=ether21 priority=0x40
add bridge=bridge1 interface=ether22
```

```
add bridge=bridge1 interface=ether23 priority=0x90
add bridge=bridge1 interface=ether24 priority=0x50
add bridge=bridge1 interface=sfp1
add bridge=bridge1 interface=ether1-master
/ip address
add address=192.168.88.1/24 comment=defconf interface=bridge1 network=\
    192.168.88.0
add address=172.18.1.4/24 interface=ether2 network=172.18.1.0
add address=192.168.181.99/24 interface=ether23 network=192.168.181.0
/ip route
add distance=1 dst-address=172.18.1.6/32 gateway=172.18.1.4
add distance=1 dst-address=192.168.181.44/32 gateway=192.168.181.99
/lcd interface
add interface=bridge1
/lcd interface pages
set 1 interfaces=ether13,ether14,ether15,ether16,ether17,ether18,ether19
/openflow port
add disabled=no interface=ether23 switch=ofswitch1
add disabled=no interface=ether21 switch=ofswitch1
add interface=ether15 switch=ofswitch1
add disabled=no interface=ether9 switch=ofswitch1
/snmp
set enabled=yes
/system lcd
set contrast=0 enabled=no port=parallel type=24x4
/system lcd page
set time disabled=yes display-time=5s
set resources disabled=yes display-time=5s
set uptime disabled=yes display-time=5s
set packets disabled=yes display-time=5s
set bits disabled=yes display-time=5s
set version disabled=yes display-time=5s
set identity disabled=yes display-time=5s
set bridge1 disabled=yes display-time=5s
set ether1-master disabled=yes display-time=5s
set ether2 disabled=yes display-time=5s
set ether3 disabled=yes display-time=5s
set ether4 disabled=yes display-time=5s
set ether5 disabled=yes display-time=5s
set ether6 disabled=yes display-time=5s
set ether7 disabled=yes display-time=5s
set ether8 disabled=yes display-time=5s
set ether9 disabled=yes display-time=5s
```

```
set ether10 disabled=yes display-time=5s
set ether11 disabled=yes display-time=5s
set ether12 disabled=yes display-time=5s
set ether13 disabled=yes display-time=5s
set ether14 disabled=yes display-time=5s
set ether15 disabled=yes display-time=5s
set ether16 disabled=yes display-time=5s
set ether17 disabled=yes display-time=5s
set ether18 disabled=yes display-time=5s
set ether19 disabled=yes display-time=5s
set ether20 disabled=yes display-time=5s
set ether21 disabled=yes display-time=5s
set ether22 disabled=yes display-time=5s
set ether23 disabled=yes display-time=5s
set ether24 disabled=yes display-time=5s
set sfp1 disabled=yes display-time=5s
/system routerboard settings
set silent-boot=no
/tool mac-server
set allowed-interface-list=none
/tool user-manager database
set db-path=user-manager
```

3.4. OpenDaylight

3.4.1. Instalación

Para instalar el `software` OpenDaylight en el controlador, el primer paso será descargarnos el material que nos ofrece su página web. Para ello, en https://www.opendaylight.org/, navegamos por `Technical Community` hacia `Getting Started for Developers` y buscamos la opción `Downloads`. [7]

Durante el estudio de esta alternativa, "Nitrogen SR2" era la versión más actualizada y estable. Una vez descargado, lo descomprimimos, navegamos hacia el directorio y lo lanzamos:

1. $ unzip karaf-0.7.2.zip
2. $ cd karaf-0.7.2/
3. $ ./bin/karaf

En la carpeta "bin" encontramos tres opciones:

- karaf: lanza Apache Karaf
- stop: para Apache Karaf
- status: devuelve el estado del servidor1

A continuación, una vez lanzado Apache Karaf, necesitaremos instalar unos módulos básicos dentro de la consola de OpenDaylight; los cuáles nos permitirán abrir la topología en el navegador, incluir ciertos nodos para la misma, subir flujos al servidor...

`opendaylight-user@root`>feature:install odl-restconf odl-l2switch odl-mdsal-apidocs odl-dlux-core odl-dluxapps-applications odl-dluxapps-topology odl-dluxapps-yangui odl-dluxapps-yangvisualizer odl-dluxapps-nodes odl-dluxapps-yangman odl-dluxapps-yangutils

Finalmente abrimos el navegador e introducimos en la caja de navegación "localhost:8181/index.html", durante nuestro primer uso nos pedirá un usuario y una contraseña: "admin" en ambos casos. Si todo ha ido correctamente, deberemos visualizar una pantalla similar a la Figura 3.2

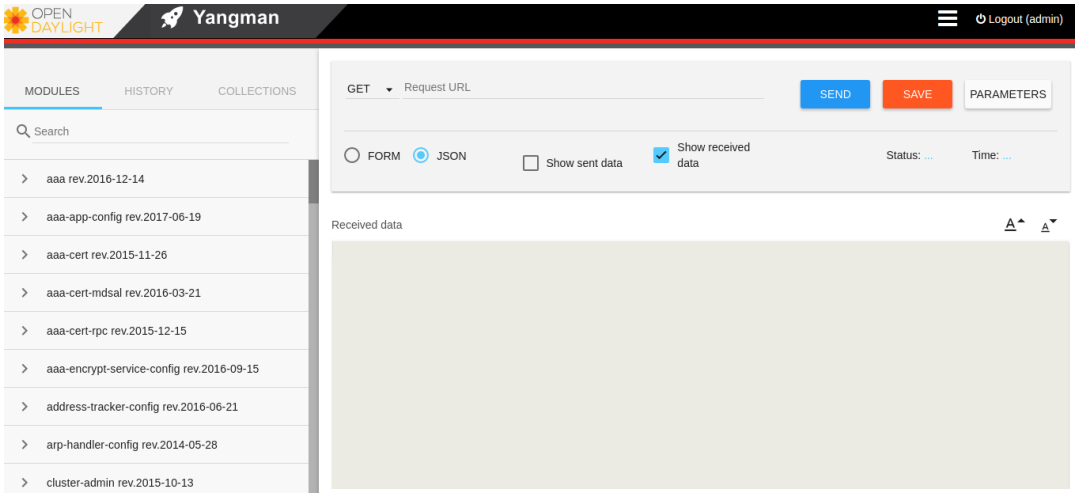

Figura 3.2: Interfaz del controlador OpenDaylight

3.4.2. Visualización de la topología

Para visualizar la topología en la propia interfaz de OpenDaylight, usaremos el archivo “topology.py” y la cargaremos en mininet (el contenido del archivo y así como la creación de una simulación de red está recogida en el Anexo A) mediante la instrucción:

$ sudo python topology.py

A continuación, probamos la conectividad de los nodos de la red a través del Protocolo de Control de Mensajes de Internet (ICMP):

`mininet>`pingall

Finalmente, navegamos hacia el apartado “topology” dentro de la interfaz OpenDaylight y tendremos una vista parecida a la de la Figura 3.3

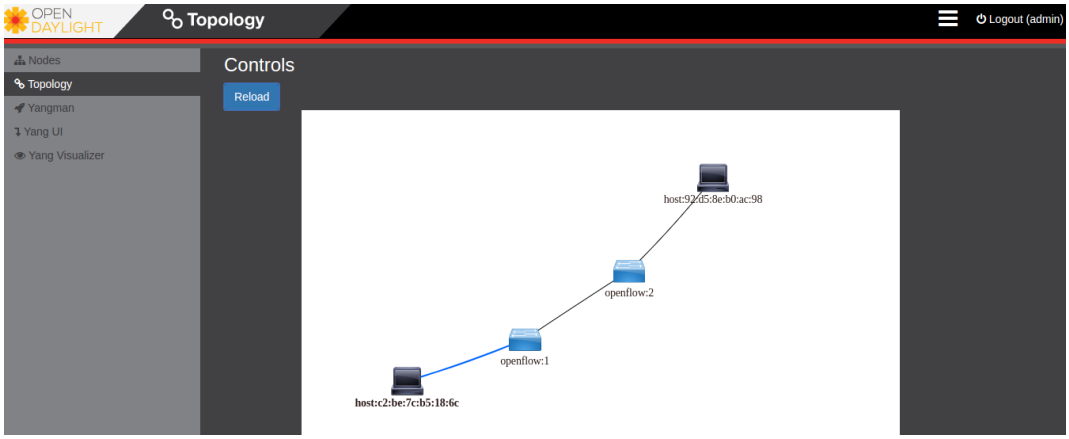

Figura 3.3: Visualización de la topología OpenDaylight

3.4.3. Inserción de flujo

Para añadir un flujo en el `software` de OpenDaylight es necesario crear un archivo xml. En este caso, vamos a utilizar uno de los archivos de ejemplo que OpenDaylight proporciona en su `wiki`. Este flujo en concreto descarta los paquetes según la dirección física (MAC) del emisor.

Es importante mencionar, que OpenDaylight no trabaja con aplicaciones, crea flujos directamente en el conmutador, y para ello usa etiquetas para cada metadato con el que puede trabajar un flujo en un conmutador:

Etiqueta XML	**Metadato en flujo**	**Descripción**
tcp-source-port	Incoming Port	Puerto entrante que registra el flujo
tcp-destination-port	Output	Puerto destino que registra el flujo
ethernet-source	Source MAC	MAC origen asociada al flujo
ethernet-destination	Destination MAC	MAC destino asociada al flujo
priority	Priority	Nivel de prioridad del flujo
vlan-id	VLAN ID	Identificador de la VLAN
idle-timeout	Idle Timeout	Tiempo de inactividad
hard-timeout	Hard Timeout	Tiempo de espera
cookie	Cookie	Identificador opaco

Tabla 3.1: Relación OpenDaylight con flujos

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
  <instructions>
    <instruction>
```

```
          <order>0</order>
          <apply-actions>
            <action>
              <order>0</order>
                  <output-action>
                    <output-node-connector>1
                         </output-node-connector>
                        <max-length>60</max-length>
                  </output-action>
                </action>
          </apply-actions>
        </instruction>
  </instructions>
  <table_id>0</table_id>
  <id>1</id>
  <cookie_mask>255</cookie_mask>
  <installHw>false</installHw>
  <match>
    <ethernet-match>
          <ethernet-source>
            <address>a6:48:47:3a:33:ef</address>
          </ethernet-source>
          <ethernet-destination>
               <address>5e:a7:e0:f9:e4:5d</address>
          </ethernet-destination>
        </ethernet-match>
  </match>
  <cookie>3</cookie>
  <flow-name>FooXf3</flow-name>
  <priority>15</priority>
  <barrier>false</barrier>
</flow>
```

A continuación, lo añadiremos como flujo al controlador mediante la orden que se muestra a continuación, donde:

- curl: Es la herramienta que podemos utilizar para hacer peticiones HTTP y transferir una URL al servidor
- –user: especifica usuario y contraseña separados por ":"
- -i: indica el encabezado del protocolo en la salida
- -X: especifica el tipo de petición:
  - PUT: Añade el flujo

- DELETE: Elimina el flujo

- -H: Manda una cabecera personalizada al servidor
- -d: directorio y archivo que contiene el flujo

$ curl –user admin:admin -i -X PUT -H “Content-Type: application/xml; charset=utf-8” -d “@/home/rewtelize/Escritorio/TFG/Controller ODL/pruebaFlows/prueba.xml” http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1

Podemos comprobar la existencia del flujo en el `switch` con la siguiente orden:

`mininet`>ssh ovs-ofctl -O OpenFlow13 dump-flows s1

Y vemos que acamos de añadir un flujo con `cookie`=0x3 tal y como se muestra en la Figura 3.4

```
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2b0000000000002, duration=1428.313s, table=0, n_packets=286, n_bytes=24310, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x3, duration=3.497s, table=0, n_packets=0, n_bytes=0, priority=15,dl_src=a6:48:47:3a:33:ef,dl_dst=5e:a7:e0:f9:e4:5d actions=output:1
 cookie=0x2b0000000000002, duration=1424.306s, table=0, n_packets=29, n_bytes=3580, priority=2,in_port=2 actions=output:1
 cookie=0x2b0000000000003, duration=1424.307s, table=0, n_packets=11, n_bytes=798, priority=2,in_port=1 actions=output:2,CONTROLLER:65535
 cookie=0x2b0000000000002, duration=1428.314s, table=0, n_packets=20, n_bytes=2487, priority=0 actions=drop
```

Figura 3.4: Flujo añadido OpenDaylight

Finalmente, comprobamos que esta norma está siendo usada por el `switch` haciendo ping entre los equipos “h1” y “h2”. Observamos así, en la Figura 3.5, que los campos “n_packets” y “n_bytes” han cambiado y que, además, ambos `hosts` no han podido establecer comunicación ICMP.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2b0000000000002, duration=1477.592s, table=0, n_packets=296, n_bytes=25160, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x3, duration=52.776s, table=0, n_packets=5, n_bytes=322, priority=15,dl_src=a6:48:47:3a:33:ef,dl_dst=5e:a7:e0:f9:e4:5d actions=output:1
 cookie=0x2b0000000000002, duration=1473.585s, table=0, n_packets=31, n_bytes=3720, priority=2,in_port=2 actions=output:1
 cookie=0x2b0000000000003, duration=1473.585s, table=0, n_packets=12, n_bytes=840, priority=2,in_port=1 actions=output:2,CONTROLLER:65535
 cookie=0x2b0000000000002, duration=1477.592s, table=0, n_packets=20, n_bytes=2487, priority=0 actions=drop
```

Figura 3.5: Flujo en acción OpenDaylight

3.5. Ryu

3.5.1. Instalación

Su instalación es bastante sencilla, basta con ingresar en la consola de comandos:

```
$ git clone git://github.com/osrg/ryu.git
$ cd ryu
$ python ./setup.py install
```

3.5.2. Visualización de la topología

1. Navegamos hacia el directorio "app" dentro del proyecto que hemos descargado:
   $ cd ./ryu/ryu/app
2. Lanzamos el servidor web con la aplicación web ya implementada.

   $ ryu run gui_topology/gui_topology.py simple_switch.py –observe-links

   - gui_topology.py lanza un servidor web permitiéndonos ver la topología. Este archivo lo trae por defecto ryu.
   - simple_machine.py es una aplicación SDN. En este apartado solo nos centramos en la visualización de la topología, en el siguiente explicaremos cómo se implementa.
   - –observe-links nos permite observar las conexiones entre los dispositivos.

3. En una nueva ventana en la consola de comandos, lanzamos mininet:
   $ sudo mn –controller=remote,ip=127.0.0.1 –topo=linear,2 –mac
4. En el navegador, abrimos la dirección 0.0.0.0:8080

Si todo ha ido bien, deberíamos tener una pantalla parecida a la de la Figura 3.6

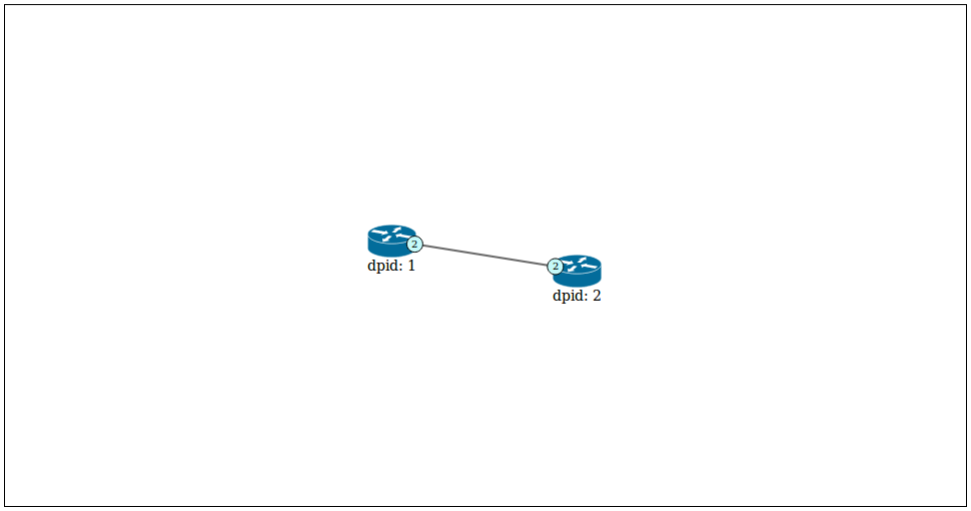

Figura 3.6: Visualización de la topología Ryu

3.5.3. Primera aplicación

Para crear la primera aplicación en Ryu [10] utilizaremos el siguiente archivo "l2" escrito en python, el cuál imbuirá el comportamiento de un `hub` en un `switch`:

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_0

class L2Switch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)

    # Llega un paquete procedente del conmutador
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        # Se desglosa el paquete en mensaje (msg), camino de datos
        # seguido (datapath) y protocolo OpenFlow tratado (ofproto)
        msg = ev.msg
        dp = msg.datapath
        ofp = dp.ofproto
```

```
        # Puerto de salida, OFPP_FLOOD (inundacion)
        actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD)]

        # Datos del paquete de salida
        out = ofp_parser.OFPPacketOut(
            datapath=dp,
            buffer_id=msg.buffer_id,
            in_port=msg.in_port,
            actions=actions)

        # El paquete es enviado al conmutador
        dp.send_msg(out)
```

Donde:

- ev.msg equivale a una estructura de datos packet_in
- msg.datapath representa un camino de datos (`switch`)
- dp.ofproto y dp.ofproto_parser trata el propotocolo OpenFlow negociado entre Ryu y el `switch`
- OFPActionOutput se usa con un mensaje packet_out y especifica un puerto de salida, en este caso, hacemos inundación (OFPP_FLOOD)
- OFPackedOut construye el mensaje packet_out
- @set_ev_cls
  - El primer argumento indica el evento que hace llamar a la función.
  - El segundo argumento indica el estado del `switch`, MAIN_DISPATCHER quiere decir que la función sólo será llamada después de completar la negociación entre Ryu y el `switch`.

Finalmente, ejecutamos la orden $ ryu-manager ./l2.py

Comprobaremos el nuevo comportamiento del `switch` a partir de la siguiente topología en mininet:

$ sudo mn –controller=remote,ip=127.0.0.1 –topo=linear,3 –mac

A continuación abriremos una consola por cada `host` y haremos ping entre h1 y h3. De esta forma, confirmaremos que el `switch` se está comportando como un `hub` ya que h2 recibe los paquetes ICMP pese a no intervenir en la comunicación. En la

elaboración de esta prueba, como se ve en la Figura 3.7, se ha usado el programa `tcpdump` para obtener datos de contenido completo.

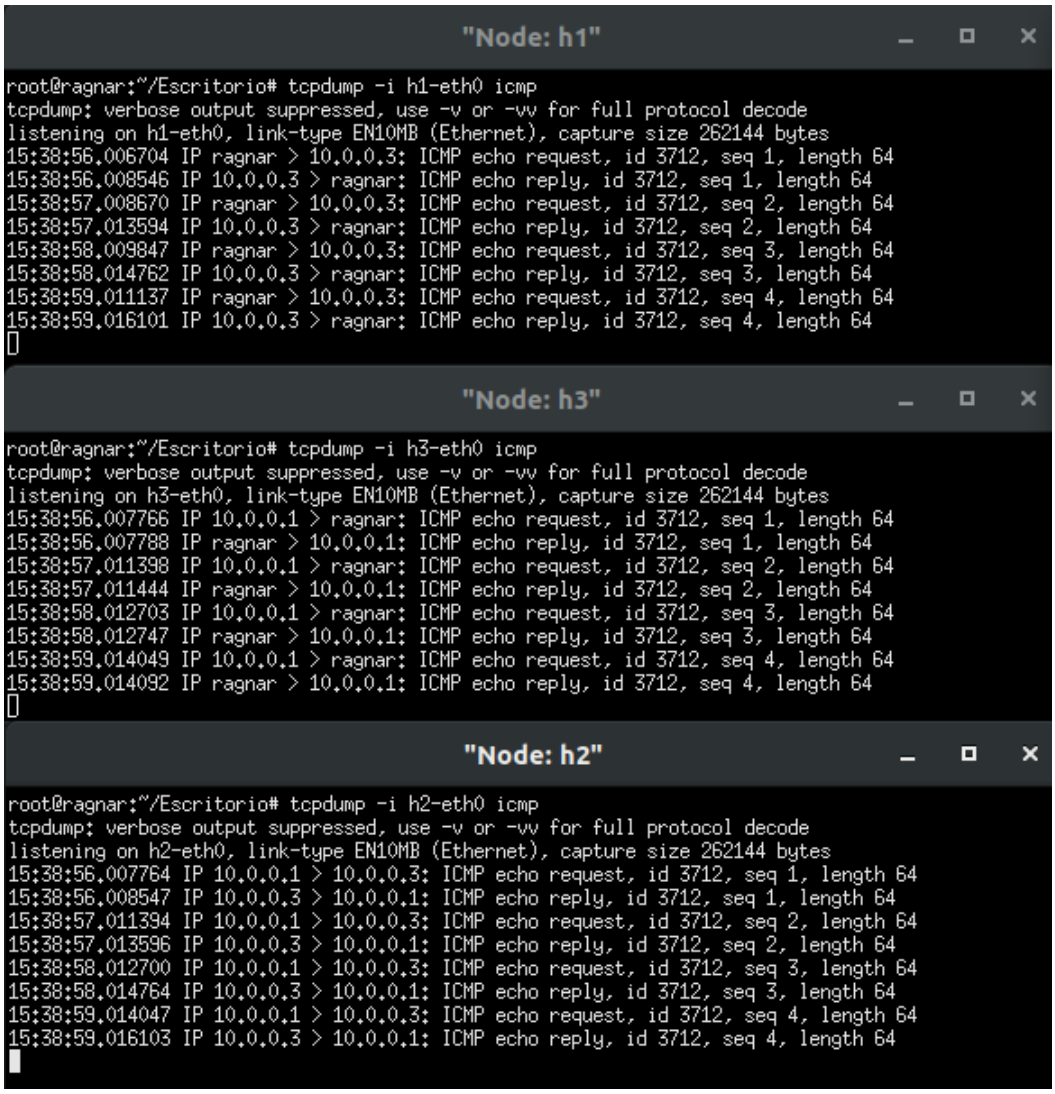

Figura 3.7: Flujo en acción Ryu

3.6. HP VAN SDN

3.6.1. Instalación

Para tener una visión más global y completa del controlador de HP, nos descargaremos un archivo ova con las funcionalidades de la aplicación Namox Spear, la cuál permite la gestión de la red sin necesidad de usar ningún tipo de lenguaje de programación. Así, tendremos que abrir el siguiente enlace: `http://tutorial.spear.narmox.com/en` y seleccionar el archivo que contiene el controlador HP VAN SDN, Mininet y Narmox Spear. [18]

Una vez tengamos preparado el servicio virtualizado y hayamos iniciado sesión en consola (el usuario y la contraseña son "narmox"), abriremos con el navegador la interfaz con la dirección del servidor; en mi caso: `https://192.168.1.139:8443/sdn/ui/Spear/app/ui/#!/dashboard`. Para abrir la interfaz, nos pedirá un usario "sdn" y la contraseña "skyline". Finalmente, si todo ha ido bien, tendremos una pantalla parecida a la de la Figura 3.8.

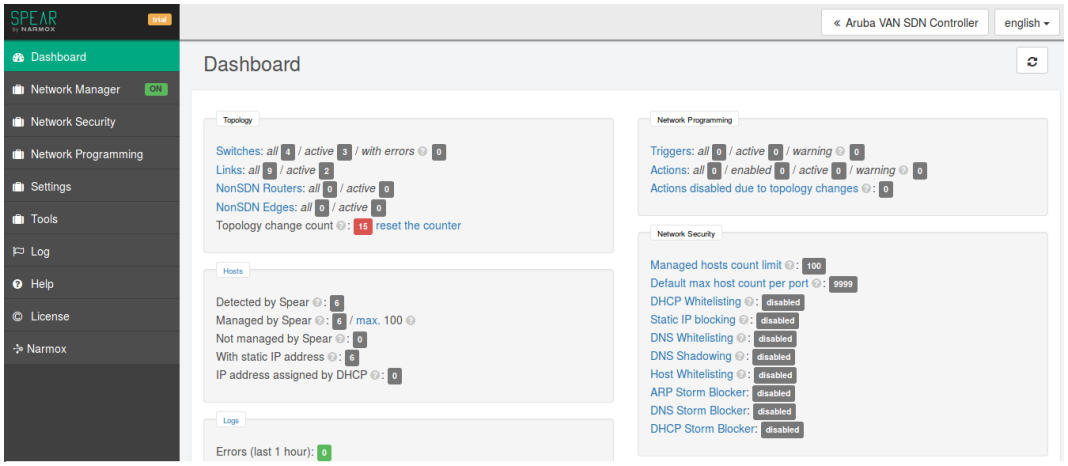

Figura 3.8: Interfaz del controlador HP VAN SDN

3.6.2. Visualización de la topología

Para visualizar la topología en la propia interfaz, lo primero será conectarnos vía SSH al servidor y crear una topología de prueba con cuatro `switches`:

```
$ ssh -X narmox@192.168.1.139
$ nx-example-1-topo-4-switches
```

A continuación, forzaremos a los nodos a encontrarse entre ellos mediante mensajes ICMP.

`mininet>`pingall

Y, por último, dentro de la interfaz nos dirigiremos al apartado “Topology” dentro de “Network Manager” tal y como muestra la siguiente imagen:

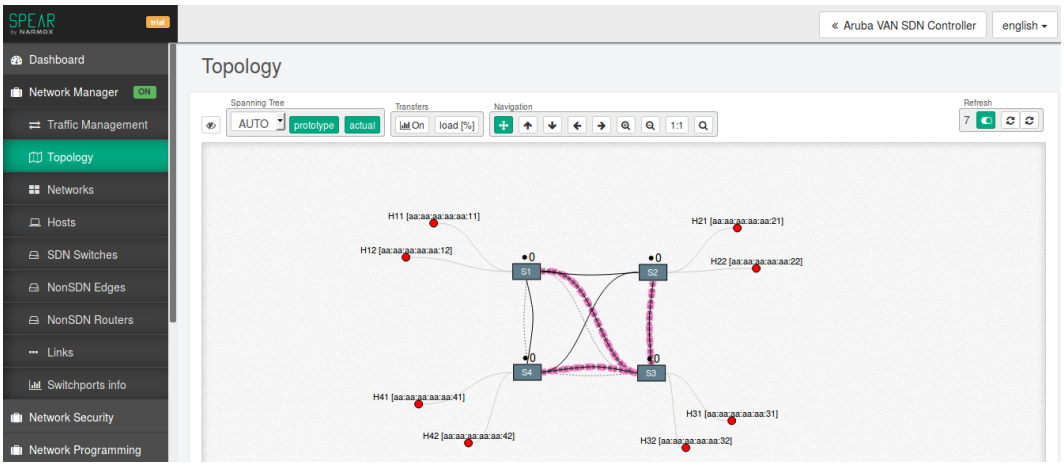

Figura 3.9: Visualización de la topología HP VAN SDN

3.6.3. Mejor camino

Podemos comprobar en todo momento qué enlaces se usan en la topología, en nuestro caso, en la Figura 3.9 observamos que los enlaces sombreados provocan que todo paquete deba pasar por el `switch` S3, es decir, el camino que recorrería un paquete del `host` H11 hasta H42 sería: H11 $\rightarrow S1 \rightarrow S3 \rightarrow S4 \rightarrow H42$.

No obstante, podemos cambiar ese comportamiendo eliminado o añadiendo determinados enlaces; teniendo en cuenta que el propio controlador siempre elegirá el mejor camino entre cada par de nodos. Podemos confirmar esto último si desactivamos el `switch` S3 con la orden:

`mininet>`switch s3 stop

Así, tal y como se muestra en la Figura 3.10, han cambiado las rutas entre los nodos y se ha quedado siempre con el mejor camino.

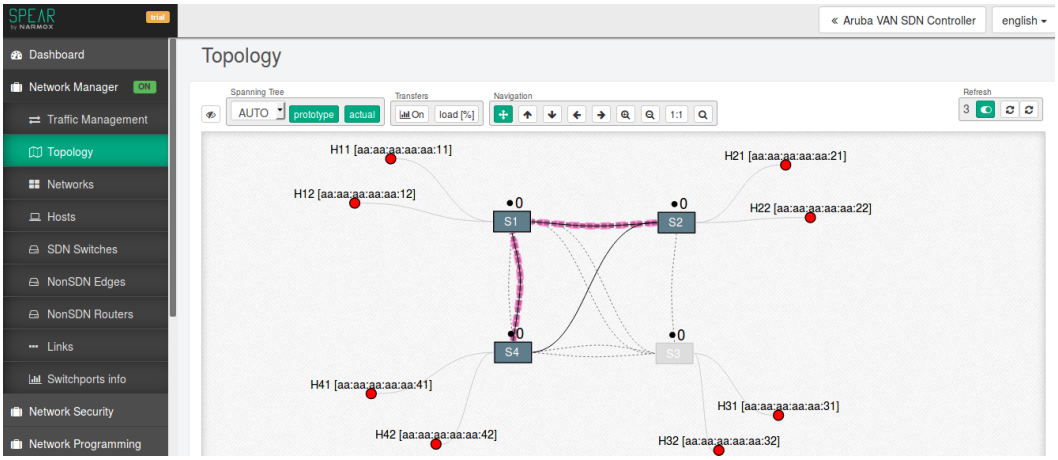

Figura 3.10: Mejor camino HP VAN SDN

3.6.4. Primer flujo

Podemos crear nuestros propios flujos si navegamos hacia el apartado “Actions” dentro de la sección “Network programming” y pulsamos New. Esto provocará que se nos abra una pequeña pantallita, en la cual tendremos que seleccionar la opción `Add flow` dentro de “type”.

En este caso de prueba, vamos a rechazar todos aquellos paquetes que tengan la dirección MAC origen de H11 y de destino H41, como se muestra en la siguiente Figura 3.11.

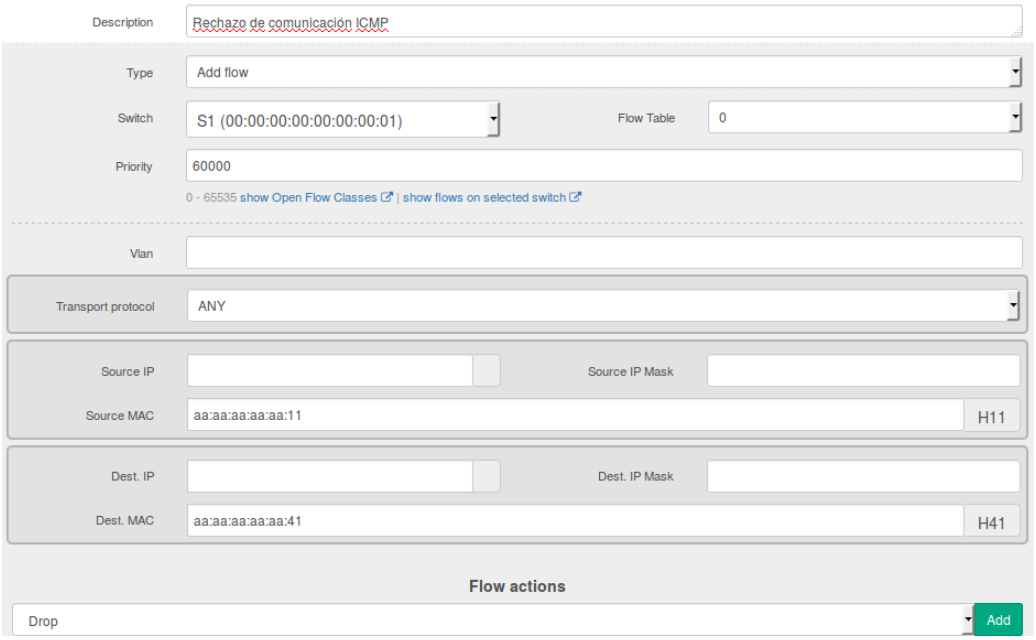

Figura 3.11: Primer flujo HP VAN SDN

Comprobaremos la eficacia de nuestro flujo si H11 puede hacer ping a H42 pero no a H41. En la Figura 3.12 observamos este comportamiento desde mininet.

```
mininet> h11 ping h42
PING 172.24.1.42 (172.24.1.42) 56(84) bytes of data.
64 bytes from 172.24.1.42: icmp_seq=1 ttl=64 time=0.722 ms
64 bytes from 172.24.1.42: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 172.24.1.42: icmp_seq=3 ttl=64 time=0.036 ms
64 bytes from 172.24.1.42: icmp_seq=4 ttl=64 time=0.048 ms
^C
--- 172.24.1.42 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.036/0.213/0.722/0.293 ms
mininet> h11 ping h41
PING 172.24.0.41 (172.24.0.41) 56(84) bytes of data.
^C
--- 172.24.0.41 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3023ms
```

Figura 3.12: Flujo en acción HP VAN SDN

3.7. Onos

3.7.1. Instalación

Para instalar ONOS, tendremos que descargarnos el archivo tar.gz (en nuestro caso, la versión más reciente es la 1.12.0) de la dirección https://wiki.onosproject.org/display/ONOS/Downloads, descomprimirlo e iniciar el servicio:

```
$ cd /opt
$ sudo wget -c http://downloads.onosproject.org/release/onos-1.12.0.tar.gz
$ sudo tar xzf onos-1.12.0.tar.gz
$ sudo mv onos-1.12.0 onos
$ /opt/onos/bin/onos-service start
```

Y, si todo ha ido bien, podremos abrir en el navegador la interfaz de la aplicación, una vez ingresemos el usuario "onos" con la contraseña "rocks" [31]. A continuación, nos aparecerá una pantalla como la de la Figura 3.13.

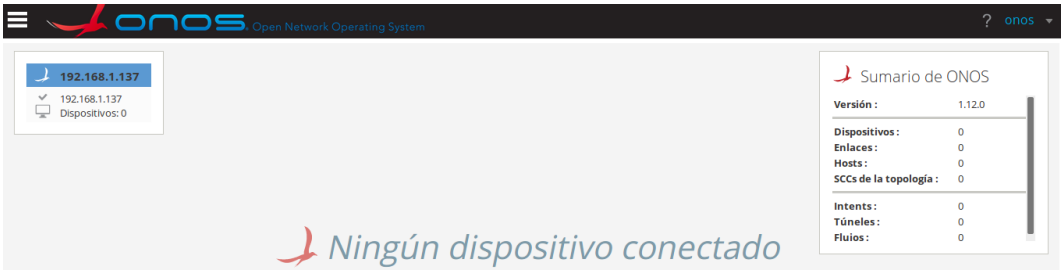

Figura 3.13: Interfaz del controlador ONOS

3.7.2. Visualización de la topología

Debido a la necesidad de instalar múltiples paquetes, vamos a descargar un servicio ova virtualizado, el cual se ofrece en https://wiki.onosproject.org/display/ONOS/Basic+ONOS+Tutorial#BasicONOSTutorial-StartMininet, configurado previamente.

Una vez instalado y corriendo, haremos doble click en el icono del escritorio `Spine Leaf Topology` para simular una red y accederemos a la sección "topology", dentro de la interfaz web. Deberíamos obtener una pantalla parecida a la de la Figura 3.14.

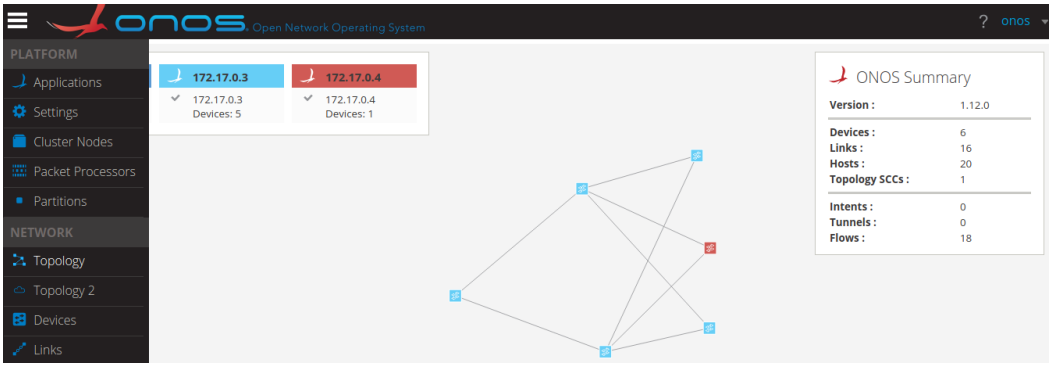

Figura 3.14: Topología del controlador ONOS

3.7.3. Primera aplicación

Si probamos a hacer ping entre los distintos hosts, comprobaremos que no son capaces de encontrarse, podemos hacer la prueba como en la Figura 3.15.

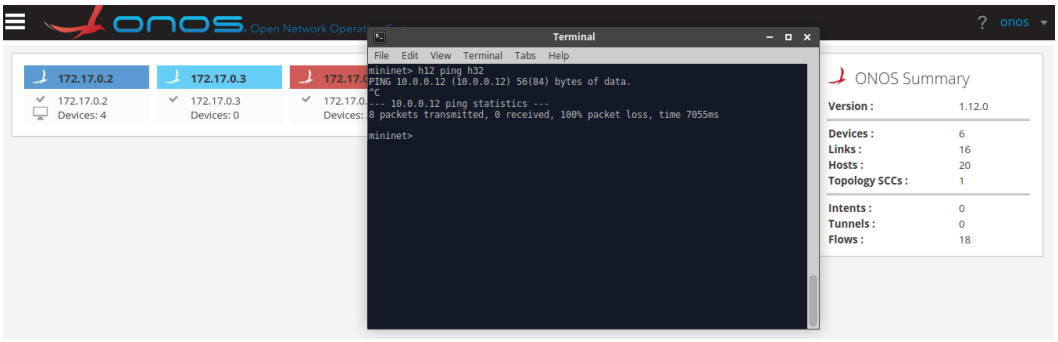

Figura 3.15: Aplicación con encaminamiento desactivado ONOS

Para que haya conectividad, necesitaremos que la aplicación `forwarding` esté activada, como se muestar en la Figura 3.16, para que los dispostivos sean capaces de encaminar los paquetes y descubrirse entre ellos.

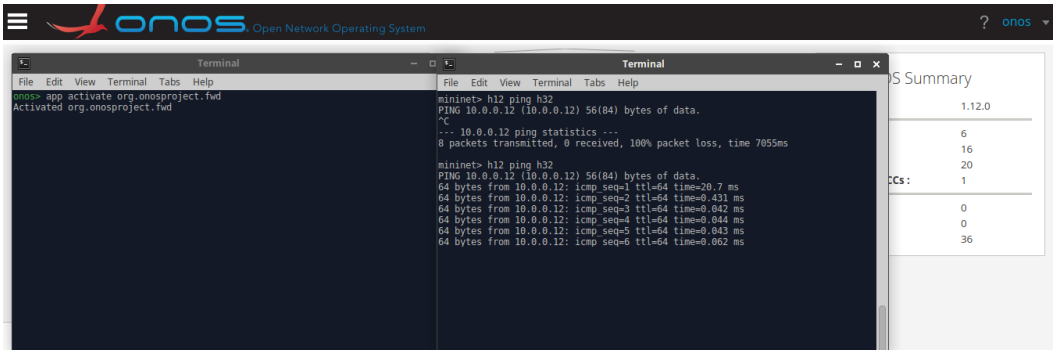

Figura 3.16: Aplicación con encaminamiento activado ONOS

Para crear una aplicación personalizada, es necesario crear el código en Java, y almacenarlo en el directorio `/tools/package/` dentro del directorio de ONOS. El código de la aplicación, que se ha usado a modo de ejemplo, se puede encontrar en su repositorio en la url: https://github.com/opennetworkinglab/onos/blob/master/apps/fwd/src/main/java/org/onosproject/fwd/ReactiveForwarding.java.

3.8. cURL

El proyecto de `software` cURL provee una biblioteca y una herramienta a utilizar por línea de comandos para la transferencia de archivos mediante numerosos protocolos.

3.8.1. Instalación

Para instalar cURL basta con ingresar en la terminal el comando de instalación y reiniciar el servidor apache para que reconozca la herramienta:

$ sudo apt-get install curl
$ sudo service apache2 restart

3.8.2. Modo de uso

Para utilizar esta herramienta, simplemente tendremos que escribir “curl” en la terminal seguidos de determinadas cláusulas:

- **curl**: transfiere una URL.
- **–user**: especifica usuario:contraseña en caso de que sea necesario un inicio de sesión.
- **-i**: incluye encabezados de protocolos en la salida.
- **-X**: especifica el tipo de petición.
  - **PUT**: Inserta.
  - **DELETE**: Elimina.
- **-H**: manda una cabecera personalizada al servidor.
- **-d**: especifica un dato o un archivo en un directorio.

De esta manera, si tenemos una instrucción como la que se muestra en el siguiente ejemplo:

$ curl –user admin:admin -i -X PUT -H “Content-Type: application/xml; charset=utf-8” -d “@/home/usuario/archivo.xml” http://127.0.0.1:8181

Estaremos mandando una cabecera “application/xml”, en una codificación UTF-8, al servidor de la dirección 127.0.0.1 relativa al archivo “archivo.xml” con un inicio de sesión previo; con usuario y contraseña “admin”.

3.9. Estimación de tamaño y esfuerzo

3.9.1. Etapa 1: Equipamiento

- Latiguillo S/FTP 1 metro ..... 5 ud.
- Latiguillo S/FTP 3 metros ..... 7 ud.
- Blade Fujitsu Primergy BX600 S2 ..... 1 ud.
- Conmutador HP 2920 ..... 2 ud.
- Conmutador Mikrotik CRS125-24G-1S-IN ..... 1 ud.
- Servidor HP ProLiant DL380 G5 ..... 1 ud.

3.9.2. Etapa 2: Personal

- Desarrollo ..... 480 h.

3.9.3. Etapa 3: Software

- Ryu SDN Framework ..... 1 licencia
- Django Web Framework ..... 1 licencia
- MySQL ..... 1 licencia

Estudio de las redes definidas por software (SDN) y desarrollo de un prototipo para la Diputación de Cádiz

Especificaciones del sistema

CLIENTE	EMPRESA PROVINCIAL DE INFORMACIÓN DE CÁDIZ, S.A. (EPICSA)
	PLAZA MADRID S/N, EDIFICIO CARRANZA, FONDO SUR, LOCAL 10, 11010 CÁDIZ
	956 26 15 00
SUMINISTRADOR	JAVIER BARROSO CANTO
	INGENIERO INFORMÁTICO
	49564855-Q
	JAVIER.BARROSOCANTO@ALUM.UCA.ES

FIRMA DEL CLIENTE **FIRMA DEL SUMINISTRADOR**

Cádiz, julio 2018

Especificaciones del sistema

Índice

4.1. Resultado de la entrevista

Con la finalidad de implementar un prototipo de SDN en un entorno real, nos reunimos con el jefe del departamento de redes de EPICSA, Manuel Añón, quien también estaba interesado en el proyecto.

Durante la entrevista realizada el 2 de Abril, detallada en la sección 3.1 Documentación de partida, se recogen unos determinados objetivos que el departamento de redes demanda para conseguir un enfoque sobre la arquitectura SDN y cómo afectaría a su red.

4.2. Objetivos

OBJ-01	**Complejidad reducida**
Descripción	El sistema deberá ofrecer una complejidad más reducida de la estructura de red

Tabla 4.1: Objetivo I: Complejidad reducida

OBJ-02	**Innovaciones y actualizaciones**
Descripción	El sistema deberá ser capaz de poder ser actualizado sin provocar impactos adversos

Tabla 4.2: Objetivo II: Innovaciones y actualizaciones

OBJ-03	**Control centralizado**
Descripción	El sistema ofrecerá un control centralizado de la estructura de red

Tabla 4.3: Objetivo III: Control centralizado

OBJ-04	**Confiabilidad y seguridad**
Descripción	El sistema será confiable y seguro, para proteger la estructura de posibles ataques maliciosos

Tabla 4.4: Objetivo IV: Confiabilidad y Seguridad

OBJ-05	**Automatización**
Descripción	El sistema debe proveer funciones automatizados para facilitar las tareas al administrador de la red

Tabla 4.5: Objetivo V: Automatización

OBJ-06	**Interfaz visualmente cómoda**
Descripción	La interfaz de la aplicación deberá ser usable e intuitiva para la gestión de la red

Tabla 4.6: Objetivo VI: Interfaz visualmente cómoda

OBJ-07	**Sistema de políticas**
Descripción	La aplicación SDN un conjunto de reglas basadas en puertos con la que rechazar paquetes.

Tabla 4.7: Objetivo VII: Sistema de políticas

OBJ-08	**Rutas de respaldo**
Descripción	El sistema ofrecerá al usuario el contenido de un servidor primario u otro de `backup` dependiendo de si el primero está operativo o no

Tabla 4.8: Objetivo VIII: Rutas de respaldo

4.3. Catálodo de requisitos del sistema

4.3.1. Requisitos de información

4.3.1.1. RI-01 Usuario

Un Usuario es una persona que ha rellenado un formulario para estar en el sistema y puede tomar acción sobre determinados requisitos funcionales.

Nombre	Descripción	Tipo
Nombre	Nombre del usuario	Texto
Apellidos	Apellidos del usuario	Texto
Correo	Dirección email del usuario	Texto
Usuario	Nombre para el inicio de sesión	Texto
Contraseña	Contraseña para el inicio de sesión	Texto
Es_admin	Tomará valor verdad cuando sea administrador	Booleano
Ultima_sesion	Fecha de la ultima sesión en el sistema	Fecha

Tabla 4.9: Atributos de la entidad Usuario

4.3.1.2. RI-02 Conmutador

Un Conmutador es un dispositivo conectado al controlador, el cuál se rige por un archivo de configuración y una aplicación.

Nombre	Descripción	Tipo
IP	IP del conmutador alcanzable por el controlador	Texto
Nombre	Nombre visible del conmutador	Texto
Versión	Versión OpenFlow con la que está trabajando	Texto
Controlador	IP del controlador al que está conectado	Texto
Instancia	Instancia asociada al conmutador	Texto
Fabricante	Fabricante del conmutador	Texto

Tabla 4.10: Atributos de la entidad Conmutador

4.3.1.3. RI-03 Configuración

Una Configuración es un archivo que contiene estructura de un conmutador. Estos serán recogidos formando un histórico y serán accesibles en todo momento.

Nombre	Descripción	Tipo
Nombre	Nombre del visible del archivo	Texto
Nota	Información extra a tener en cuenta	Texto
Archivo	Nombre del archivo de configuración	Texto
Fecha	Fecha de creación	Fecha

Tabla 4.11: Atributos de la entidad Configuración

4.3.1.4. RI-04 Aplicación

Una Aplicación es un archivo que contiene el comportamiento que un dispositvo que soporte OpenFlow puede imbuir en su sistema.

Nombre	Descripción	Tipo
Nombre	Nombre visible de la aplicación	Texto
Descripción	Información sobre el comportamiento que imbuye	Texto
Archivo	Nombre de la aplicación	Texto
Fecha	Fecha de creación	Fecha
Autor	Persona u organización que ha creado el archivo	Texto

Tabla 4.12: Atributos de la entidad Aplicación

4.3.1.5. RI-05 Política

Una Política es una relación entre dos puertos y una acción, que define un relación de paquetes a aceptar o rechazar entre dos equipos de un conmutador.

Nombre	Descripción	Tipo
Número	Número visible de la política	Numérico
Switch	Direccion del dispositivo	Numérico
Origen	Puerto origen de la política	Numérico
Destino	Puerto destino de la política	Numérico
Acción	Aceptar/Rechazar paquete	Texto

Tabla 4.13: Atributos de la entidad Política

4.3.1.6. RI-06 Gráfico

Un Gráfico es una representación visual de la carga de los conmutadores.

Nombre	Descripción	Tipo
Fecha	Fecha del gráfico	Fecha
IP	IP del conmutador	Texto
Tráfico	Cantidad de bytes transmitidos	Numérico
Tipo	Tráfico de entrada o de salida	Texto

Tabla 4.14: Atributos de la entidad Política

4.3.2. Requisitos funcionales

4.3.2.1. Diagrama de casos de uso

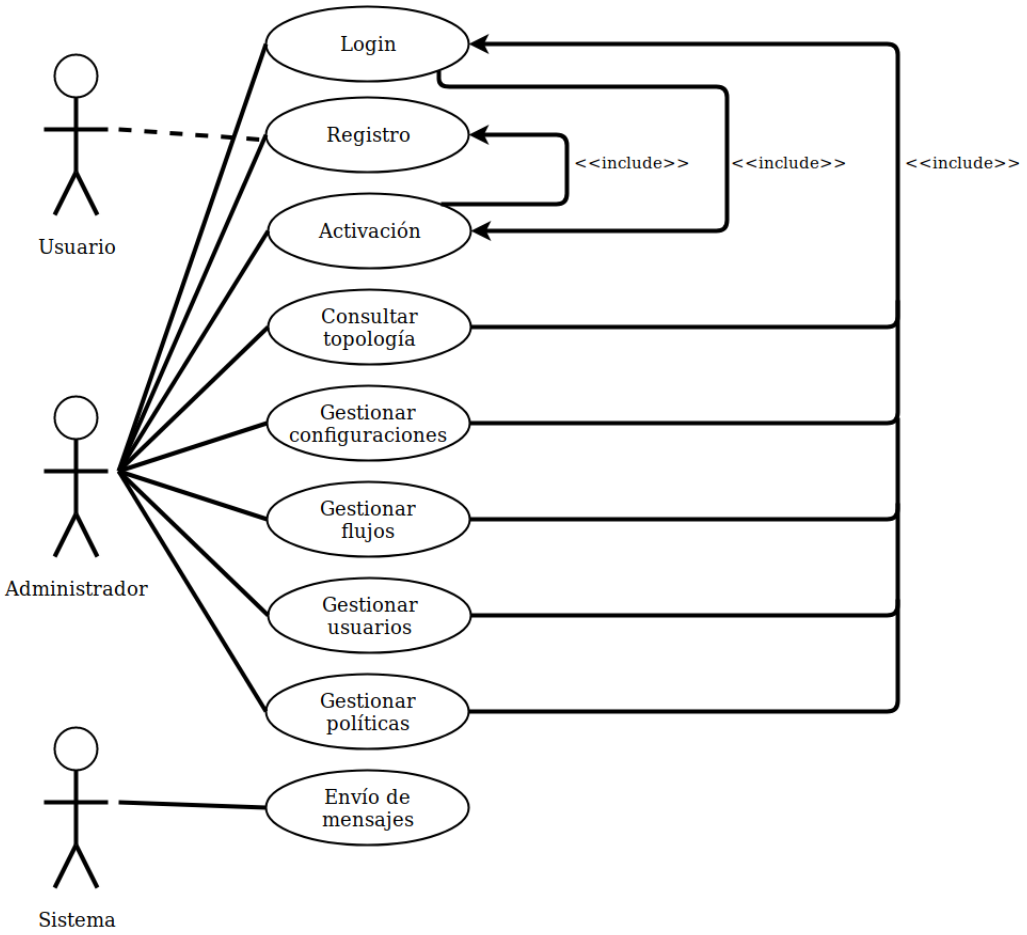

Figura 4.1: Diagrama de casos de uso

4.3.2.2. Definición de actores

En la aplicación web van a intervenir tres actores:

- **Usuario:** Un usuario es cualquier trabajador de EPICSA que quiera registrarse en la aplicación web.

- **Administrador:** Un administrador es un usuario al que ha sido validado su registro y, por tanto, posee los rivilegios para iniciar sesión en la aplicación web, validad a otros usuarios y gestionar el resto de activos disponibles en la interfaz.

- **Sistema:** De manera automatizada, el sistema puede mandar correos en caso de que fuera necesario.

4.3.2.3. Requisitos

Requisitos	Descripción
RF-01 Registro	El usuario podrá darse de alta.
RF-02 Activación	Los administradores activarán las cuentas de los nuevos usuarios. **Anotación:** <include>Registro.
RF-03 Login	Los administradores deberán iniciar sesión en la aplicación para poder trabajar con ella. **Anotación:** <include>Activación.
RF-04 Visualizar dispositivos	El administrador podrá visualizar las características de los nodos que conformen la red. **Anotación:** <include>Login.
RF-05 Gestionar configuraciones	El administrador tendrá a su disposición un historial con las configuraciones de los equipos. **Anotación:** <include>Login.
RF-06 Gestionar flujos	El administrador podrá crear, actualizar, borrar y modificar un determinado flujo. **Anotación:** <include>Login.
RF-07 Gestionar usuarios	El administrador podrá actualizar, borrar y modificar toda información sobre un determinado usuario. **Anotación:** <include>Login.
RF-08 Gestionar políticas	El administrador podrá actualizar, borrar y modificar una política de comunicación. **Anotación:** <include>Login.
RF-09 Envío de mensajes	El sistema, de manera automatizada, mandará correos en determinadas circunstancias definidas por los usuarios. **Anotación:** <include>Login.

Tabla 4.15: Requisitos funcionales

4.3.3. Requisitos no funcionales

- **RNF01 - Adecuación funcional**: Existen funciones y propiedades específicas en la aplicación que consiguen el correcto cumplimiento de los requisitos funcionales.

- **RNF02 - Rendimiento**: El programa responde con aprovechamiento y acierto bajo determinadas circunstancias; es decir, reduce el tiempo de latencia al mínimo en escenarios complejos.

- **RNF03 - Portabilidad**: La aplicación tendrá la capacidad de transferirse a diferentes entornos. Una buena forma de asegurarse esta característica inherente en el sistema es mediante la utilización de `software` libre, ya que el código es compatible con todos los sistemas operativos.

- **RNF04 - Compatibilidad**: Al ser `software` libre, su coexistencia, así como su integración con otros entornos; resulta más sencilla, ya que tenemos acceso al código y la opción de modificarlo o añadir nuevos complementos para conseguir esa interacción.

- **RNF05 - Usabilidad**: El `software` proporciona una interfaz visualmente cómoda para la gestión y visualización de la red.

- **RNF06 - Fiabilidad**: El programa mantiene una cierta capacidad de desempeño en condiciones y tiempo determinadas.

- **RNF07 - Seguridad**: La interfaz tiene implementado un sistema de autentificación para evitar accesos no autorizados. Además, un usuario con rol "administrador", tendrá la opción de registrar políticas de uso sobre los dispositivos de la red.

- **RNF08 - Mantenibilidad**: Una implementación mediante `software` libre facilita su capacidad para ser modificado en un futuro en nuevas demandas venideras.

4.4. Diseño conceptual

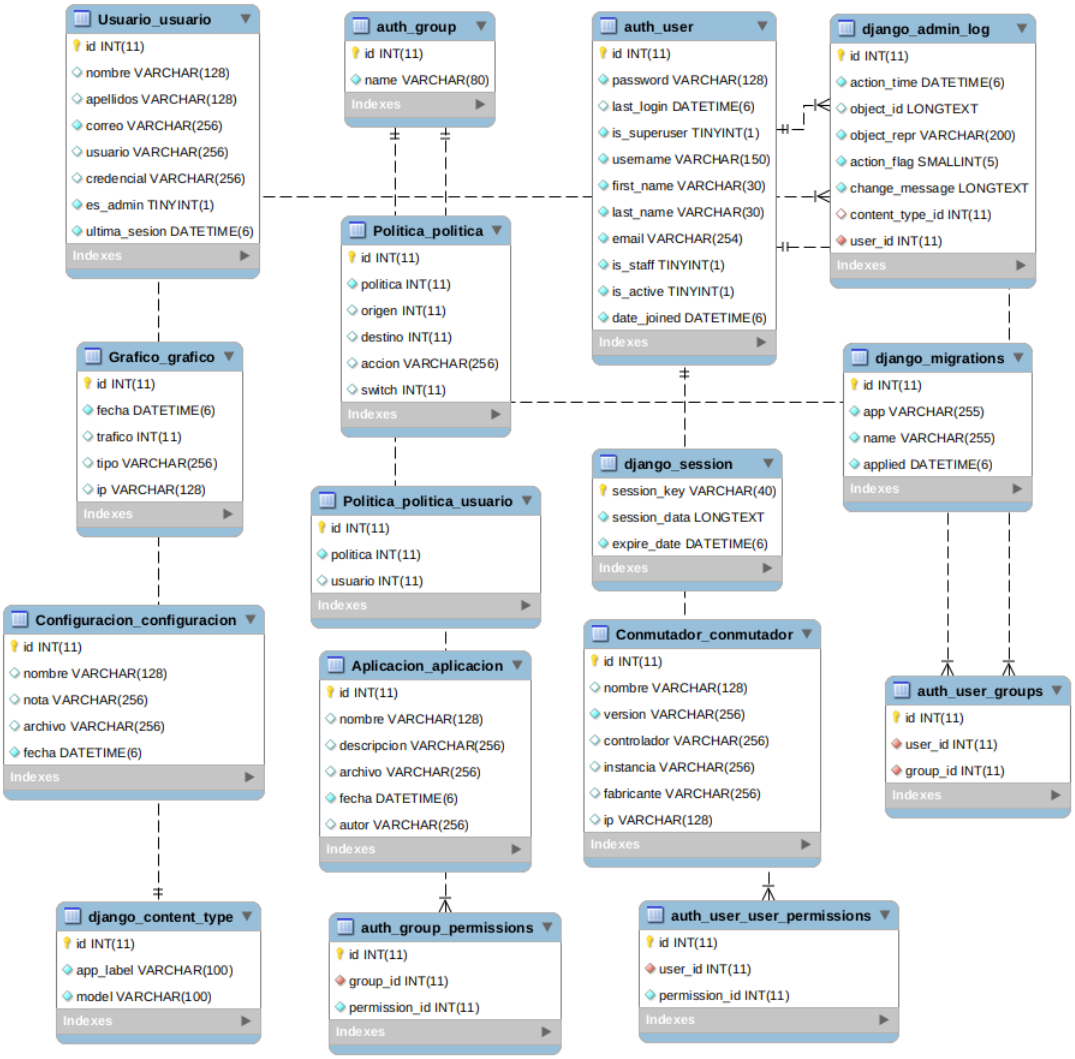

Figura 4.2: Modelo Entidad/Relación

4.5. Matriz de rastreabilidad de requisitos

Se muestra, en la Tabla 4.16, la matriz de rastreabilidad de los requisitos funcionales, junto con los objetivos del sistema demandados en la entrevista con el cliente; según su codificación.

	RF-01	RF-02	RF-03	RF-04	RF-05	RF-06	RF-07	RF-08	RF-09
OBJ-01				↵	↵	↵			
OBJ-02					↵	↵			
OBJ-03					↵	↵			
OBJ-04	↵	↵	↵					↵	↵
OBJ-05									↵
OBJ-06				↵	↵	↵	↵	↵	
OBJ-07								↵	
OBJ-08						↵			

Tabla 4.16: Matriz de rastreabilidad

Estudio de las redes definidas por software (SDN) y desarrollo de un prototipo para la Diputación de Cádiz

Presupuestos

CLIENTE	EMPRESA PROVINCIAL DE INFORMACIÓN DE CÁDIZ, S.A. (EPICSA)
	PLAZA MADRID S/N, EDIFICIO CARRANZA, FONDO SUR, LOCAL 10, 11010 CÁDIZ
	956 26 15 00
SUMINISTRADOR	JAVIER BARROSO CANTO
	INGENIERO INFORMÁTICO
	49564855-Q
	JAVIER.BARROSOCANTO@ALUM.UCA.ES

FIRMA DEL CLIENTE **FIRMA DEL SUMINISTRADOR**

Cádiz, julio 2018

Presupuestos

Índice

5.1. Etapa 1: Equipamiento

Componente	**Detalles**	**Precio Unitario (€)**	**Cantidad**	**Total (€)**
Latiguillo S/FTP 1 metro	Precio/unidad	1,70	5	8,50
Latiguillo S/FTP 3 metros	Precio/unidad	3,30	7	23,10
Blade Fujitsu *Primergy BX600 S2*	Precio/unidad	9.200,18	1	9.200,18
Conmutador HP 2920	Precio/unidad	1.114,59	2	2.229,18
Conmutador Mikrotik *CRS125-24G-1S-IN*	Precio/unidad	160,49	1	160,49
Servidor HP ProLiant *DL380 G5*	Precio/unidad	751,25	1	751,25
			Total sin IVA: 12.372,70 €	
			Total con IVA: 14.970,97 €	

Tabla 5.1: Presupuesto etapa 1: Equipamiento

5.2. Etapa 2: Personal

Componente	**Detalles**	**Precio Unitario (€)**	**Tiempo (h)**	**Total (€)**
Autor del proyecto	Precio/hora	23	480	11.040
			Total: 11.040 €	

Tabla 5.2: Presupuesto etapa 2: Personal

5.3. Etapa 3: Software

Componente	**Detalles**	**Precio Unitario (€)**	**Licencias**	**Total (€)**
Ryu SDN Framework	Precio/licencia	0	1	0
Django Web Framework	Precio/licencia	0	1	0
MySQL	Precio/licencia	0	1	0
				Total: 0 €

Tabla 5.3: Presupuesto etapa 3: Software

5.4. Resumen final del presupuesto

Etapas	**Precio total (€)**
Etapa 1: Equipamiento	14.970,97
Etapa 2: Personal	11.040
Etapa 3: Software	0
	Total: 26.010,97 €

Tabla 5.4: Resumen final del presupuesto